Sistema avanzado de predicción de crisis migrañosas

Kevin Henares Vilaboa

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería Informática

26 de febrero de 2018

Convocatoria: febrero de 2018

Calificación obtenida: 9.5

Directores:

José Luis Risco Martín
Josué Pagán Ortiz

Índice general

Índice de figuras

Autorización de difusión

Kevin Henares Vilaboa

26 de febrero de 2018

El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Sistema avanzado de predicción de crisis migrañosas", realizado durante el curso académico 2017-2018 bajo la dirección de José Luis Risco Martín y con la colaboración externa de dirección de Josué Pagán Ortiz en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

La migraña es una de las enfermedades neurológicas más incapacitantes y está presente en alrededor del 10 % de la población mundial. Sin embargo, aunque se han hecho avances en cuanto a tratamientos de prevención, aún no se ha conseguido crear una cura para esta enfermedad.

Por otra parte, la medicación existente para neutralizar episodios de migraña presenta una característica destacable: es notablemente más efectiva cuando se toma al inicio del episodio. Por el contrario, si se administra cuando ya haya comenzado el dolor su efecto queda reducido (o incluso anulado). No obstante, la mayoría de pacientes de migraña no son capaces de predecir cuando va a comenzar el dolor en un episodio de migraña, por lo que suelen realizar un uso no óptimo de la medicación asociada.

Este proyecto presenta un sistema capaz de predecir la fases de dolor en episodios de migraña. Para ello, se recogen datos de pacientes y se crean modelos predictivos que permiten estimar la probabilidad de aparición de una nueva etapa de dolor. Estas predicciones permiten la generación de alarmas con una antelación de hasta 45 minutos. Por tanto, posibilita un uso adecuado de los medicamentos y aumenta la eficacia de los mismos.

Palabras clave

Migraña, modelos predictivos, series temporales, simulación de eventos discretos, FPGA, VHDL.

Abstract

Migraine is one of the most disabling neurological diseases and is present in around 10 % of the world population. However, although progress has been made in terms of prevention treatments, a cure for this disease has not yet been achieved.

On the other hand, the existing medication to neutralize migraine episodes presents a remarkable feature: it is quite more effective when taken at the beginning of the episode. On the contrary, if it is administered when the pain has already begun its effect is reduced (or even canceled). However, most migraine patients are not able to predict when the pain will begin in a migraine episode, so they usually make a non-optimal use of the associated medication.

This project presents a system capable of predicting pain phases in episodes of migraine. For this purpose, patient data is collected and predictive models are created to estimate the probability of appearance of a new pain stage. These predictions allow the generation of alarms up to 45 minutes in advance. Therefore it makes possible an adequate use of the medicines, increasing its effectiveness.

Keywords

Migraine, predictive models, time series, discrete events simulation, FPGA, VHDL.

Agradecimientos

A José Luis Risco Martín y Josué Pagán Ortiz, por su apoyo y colaboración durante la realización de este proyecto.

Capítulo 1

Introducción

1.1. Motivación

La migraña es un tipo de enfermedad neurológica consistente en dolores de cabeza recurrentes, de varias horas de duración (usualmente de 2 a 72 horas)[16] Afecta en torno al 10 % de la población mundial y es considerada una de las enfermedades neurológicas más incapacitantes. Debido a influencias hormonales, afecta en mayor medida a la población femenina (siendo el doble de frecuente). Suele aparecer en la pubertad y concentra su impacto en la franja de edad de los 35 a los 45 años (repercutiendo a menudo de por vida).

Las frecuencia de los episodios es muy variable, situándose la media en 1,5 episodios mensuales[25]. En casos de migraña crónica, ésta puede estar presente en más de 15 días cada mes[4].

Actualmente no existe una cura para la migraña, aunque hay diversos tratamientos preventivos que pueden reducir el número de episodios hasta a la mitad. Sin embargo, en el caso de pacientes con crisis agudas se tiende a la sobremedicación, aumentando así los costes y efectos secundarios[9] (siendo el coste anual medio por paciente de 1222 euros).

Por otra parte, la medicación asociada es considerablemente más efectiva si se administra con antelación al inicio del episodio. Se estima que tomar el medicamento con antelación suficiente elimina el 66 % de los episodios y reduce la intensidad del 33 % de los restantes[26]. Por el contrario, si se toma una vez se ha iniciado el dolor su efecto queda reducido o incluso

anulado.

A parte de la fase de dolor (etapa más conocida habitualmente), la migraña va acompañada de una cascada de efectos neurológicos y habitualmente está precedida por otros síntomas (pródromos y aura). Además, al finalizar el episodio de dolor y con posterioridad al mismo aparecen nuevos efectos (síntomas postdrómicos)[26]. Estas fases se encuentran detalladas en el apartado 2.1.

1.2. Objetivos

El objetivo de este proyecto es la implementación y validación de un sistema que sea capaz de predecir episodios de dolor con antelación suficiente para que los pacientes consigan anular o reducir notablemente la intensidad de sus episodios de migraña. Para ello, se desarrollará un sistema hardware que tenga como entradas las principales variables biomédicas asociadas con esta enfermedad (temperatura corporal, sudoración, ritmo cardíaco y saturación de oxígeno). En base a ellas, y usando modelos entrenados previamente, se evaluará la posibilidad de que aparezcan nuevos episodios de dolor.

Para llegar a la obtención de este sistema aparecen las siguientes tareas dependientes:

- Extracción de datos de pacientes de forma ambulatoria.
- Entrenamiento y validación de modelos predictivos.
- Representación del sistema en un formalismo que permita simular el comportamiento del sistema y respaldar la viabilidad de su desarrollo en un entorno real.
- Obtención de sensores que permitan proporcionar las entradas citadas al sistema.
- Implementación del sistema en una FPGA mediante un lenguaje de especificación hardware.
- Simulación de cada uno de los componentes que conforman el sistema, de forma que se compruebe el correcto funcionamiento de los mismos.

- Desarrollo de una placa que permita agrupar las distintas entradas y añadir filtros que estabilicen las señales de entrada.
- Generación de un entorno que permita validar las funcionalidades clave del sistema.
- Validación de la implementación realizada directamente sobre el dispositivo utilizado.

Capítulo 2

Visión general

En esta sección se introducen los conceptos que se usan a lo largo del proyecto. Se empieza hablando de la migraña, comentando sus características, los segmentos de población afectados y sus síntomas asociados. Se continúa hablando de los modelos usados para la estimación de valores de entrada ante la aparición de caídas de sensores (ARX) y la predicción del dolor asociado a la migraña ($N4SID$). Posteriormente se introduce el sistema de modelado *DEVS*, que permite la validación formal del sistema a desarrollar. Por último se comentan los procesos de desarrollo y protocolos de comunicación hardware que influyen en el desarrollo del sistema.

2.1. Fases de la migraña

Como se ha comentado anteriormente, la migraña tiene otros efectos menos conocidos que están presentes en gran cantidad de pacientes. Estos son los siguientes:

- **Pródromos**: síntomas de diversa índole (cognitivos, físicos, sensoriales, etc.) que ocurren entre 2 y 48 horas antes del dolor[19]. A pesar de su relación con los episodios de dolor, estos síntomas no pueden ser usados para predecir el dolor (ya que son subjetivos y fácilmente confundibles con estados cotidianos independientes de la migraña). Algunos de los más comunes son: cansancio, somnolencia, hiperactividad, bostezos o rigidez nucal y están presentes en el 80 % de los pacientes[7].

- **Aura**: síntomas que aparecen hasta una hora antes del dolor y de características variadas, siendo las más comunes alteraciones visuales (zona ciega, aumento de la sensibilidad a la luz, etc.), alteraciones del lenguaje y sensación de hormigueo. Se estima que está presente en un 10-20 % de los pacientes[22][24].

- **Síntomas postdrómicos**: frecuentemente, tras la fase de dolor, permanecen ciertos síntomas relacionados con la migraña. Algunos ejemplos serían: disminución de la concentración, cansancio, somnolencia, etc.

La figura 2.1 muestra como se entrelazan las diferentes fases de la migraña. La línea verde representa los síntomas prodrómicos, que aparecen al principio del episodio y pueden extenderse hasta la fase de dolor (pudiendo llegar a entrelazarse). La línea amarilla representa los episodios de aura, presentes poco antes del comienzo del dolor y de poca duración. La línea roja representa la fase de dolor, etapa principal y más conocida de las migrañas. En esta etapa también es usual la aparición de síntomas concomitantes, como pueden ser mareos, náuseas, fotofobia, sonofobia, osmofobia, etc. Por último, la línea azul representa los síntomas postdrómicos, presentes al final del episodio.

Figura 2.1: Fases de una migraña

Por último, es importante mencionar que existen multitud de factores desencadenantes que pueden inducir un episodio de migraña. Estos varían ampliamente entre pacientes y pueden ser de distinta índole, encontrándose entre ellos factores ambientales, cambios hormonales, ingesta de productos alimenticios, etc.[2]. Además, tras la exposición a estos factores

el inicio del episodio provocado puede demorarse hasta 24 horas.

2.2. Modelos

2.2.1. Subspace State Space System Identification (N4SID)

N4SID es un algoritmo basado en un modelo de espacio de estados. Dado que los algoritmos N4SID son no-iterativos se evitan inconvenientes como convergencias no garantizadas o sensibilidad a las estimaciones iniciales típicos de los algoritmos iterativos.

En él se definen estados para describir las ecuaciones diferenciales que calculan entradas presentes y futuras en base a las entradas pasadas y presentes. Se define formalmente con la siguiente ecuación:

$$\begin{aligned} x_{[k]+1} &= A_{X_{[k]}} + BU_{[k]} + w_{[k]} \\ y_{[k]} &= C_{x_{[k]}} + D_{u_{[k]}} + v_{[k]} \end{aligned} \tag{2.1}$$

donde:

- $U_{[k]}$ son las entradas (U) en tiempo k.
- $y_{[k]}$ son las salidas (Y) en tiempo k.
- A es la matriz de transición de estados.
- B relaciona los estados en un tiempo k ($x_{[k]}$) con las entradas.
- C es el estado en la matriz de salida.
- $v_{[k]}$ y $w_{[k]}$ representan ruido blanco (inmensurable).

2.2.2. Auto Regressive models with Exogeneous inputs (ARX)

Los modelos AR, o modelos autoregresivos, asumen que los valores de la variable de salida dependen de los valores que haya tomado esa variable en el pasado.

Los modelos ARX son modelos autoregresivos en los que el valor de salida no solo depende de los valores pasados de la misma variable, sino que también se ve influido por los valores pasados de un conjunto de variables exógenas (de origen externo).

Formalmente se describe de la siguiente forma:

$$\begin{aligned} y(t) + a_1 * y(t-1) + ... + a_{n_a} * y(t-n_a) = \\ b_1 * u(t-n_{[k]}) + ... + b_{n_b} * u(t-n_b-n_{[k]}+1) + e(t) \end{aligned} \tag{2.2}$$

donde:

- $y(t)$ salida en tiempo t.
- n_a es el número de polos.
- n_b es el número de ceros más uno.
- $n_{[k]}$ es el número de muestras de entrada que se contabilizan previas a la entrada actual del sistema.
- $y(t-1)...y(t-n_a)$ son las salidas previas de las que depende la salida actual.
- $u(t-n_{[k]})...u(t-n_{[k]}-n_b+1)$ son las entradas previas de las que depende la salida actual.
- e(t) representa un ruido blanco.

2.3. Discrete Event System Specification (DEVS)

DEVS es un formalismo para la descripción de sistemas en base a un comportamiento modular y jerárquico. *DEVS* es universal y único en cuanto al modelado de sistemas de eventos discretos, de forma que cualquier sistema que acepte eventos como entradas y genere eventos como salidas puede ser representado como un modelo *DEVS*[28].

Para la descripción de un sistema *DEVS* se usa el concepto de componente, distinguiendo entre dos tipos diferentes: atómicos y acoplados.

2.3.1. Componentes atómicos

Un componente atómico es la entidad mínima en *DEVS*. En ella, en base al estado actual, se reciben y procesan eventos de entrada y se generan eventos de salida (pudiendo cambiar el estado en este procesamiento). De esta forma, un componente atómico en *DEVS* está definido por tres conjuntos (entradas, salidas y estados) y cinco funciones (avance temporal, transición externa, transición interna, confluente y salida). Formalmente, se representa de la siguiente forma:

$$A = \langle I, O, X, S, Y, \lambda, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, ta \rangle \tag{2.3}$$

- I es el conjunto de puertos de entrada.
- O es el conjunto de puertos de salida.
- X es el conjunto de entradas, descritas como pares puerto-valor: $\{p, v\}$.
- S es el espacio de estados. Incluye el estado actual y dos parámetros adicionales: σ y *phase*. σ es el tiempo que debe pasar para que se produzca el siguiente evento, mientras que *phase* describe el estado actual (en lenguaje natural).
- Y es el conjunto de salidas, también descritas como pares puerto-valor: $\{p, v\}$.
- $\lambda : S \to Y$ es la función de salida. Cuando el tiempo pasado desde la última función de salida coincide con σ se ejecuta la función λ.
- $\delta_{int} : S \to S$ es la función de transición interna. En ella se cambia el estado S, *phase* y σ. Se ejecuta inmediatamente después de la función de salida (λ).
- $\delta_{ext} : Q \cdot X^b \to S$ es la función de transición externa. Se ejecuta automáticamente cada vez que se produce un evento externo en cualquiera de los puertos de entrada, cambiando el estado actual si es necesario.

- $Q = (s, e), s \in S, 0 \leq e \leq ta(s)$ es el conjunto completo de estados, donde e es el tiempo transcurrido desde la última transición.
- X^b es un subconjunto de elementos de X.

- $\delta_{con} : Q \cdot X^b \rightarrow S$ es la función de confluencia, sujeta a $\delta_{con}(s, \emptyset) = \delta_{int}(s)$. Esta transición aplica cuando δ_{ext} y δ_{int} deben ser ejecutadas en el mismo instante de tiempo.

- $ta(s) : S \rightarrow \Re_0^+ \cup \infty$ es la función de avance de tiempo.

2.3.2. Componentes acoplados

Un componente acoplado agrega y conecta dos o más componentes (atómicos o acoplados). De esta forma, los componentes se organizan de manera jerárquica hasta definir el sistema completo. Formalmente se describe como:

$$M = \langle I, O, X, Y, C_i, EIC, EOC, IC \rangle \tag{2.4}$$

donde:

- I es el conjunto de puertos de entrada (externos).
- O es el conjunto de puertos de salida (externos).
- X es el conjunto de eventos de entrada externos.
- Y es el conjunto de eventos de salida.
- C_i es un conjunto de componentes *DEVS* (atómicos o acoplados).
- EIC es la relación de acoplamiento de entrada. Define como se conectan los puertos de entrada con los componentes internos.
- EOC es la relación de acoplamiento de salida. Define como se conectan los puertos de salida con los compentes internos.

- *IC* es la relación de acoplamiento interna. Define como se conectan los componentes internos entre sí.

2.4. Hardware

2.4.1. Síntesis

Las *FPGAs* (Field Programmable Gate Arrays) son dispositivos cuya estructura interna puede ser reconfigurada usando lenguajes de descripción hardware (*HDL*, Hardware Description Languages). Están formadas por bloques lógicos, cada uno de los cuales está formado por células lógicas compuestas de circuitos básicos (como sumadores, flip-flops o multiplexores). Las conexiones entre los distintos bloques lógicos pueden ser alteradas, de forma que se altere su estructura y funcionalidad.

Los lenguajes de descripción hardware, como *VHDL* (VHSIC Hardware Description Language, siendo VHSIC el acrónimo de Very High Speed Integrated Circuit) o *Verilog*, estructuran el comportamiento del sistema en módulos (definiendo sus entradas, salidas y modo de operación). Estos módulos pueden ser replicados y combinados de forma que se creen componentes cada vez más complejos, hasta llegar al sistema final a desarrollar.

Una vez se ha desarrollado el sistema en un *HDL*, se deberá pasar por varias fases antes de poder cargarlo en la FPGA:

- **Síntesis**: proceso en el que el código *HDL* se convierte a una descripción lógica de bajo nivel. Por tanto, se determinan los recursos que serán necesarios para la construcción del circuito y como se deben conectar.

- **Implementación**: proceso en el que se adapta el diseño sintetizado a las características de la FPGA que se quiera usar.

- **Generación del bitstream**, archivo que podrá ser utilizado para cargar el sistema en la FPGA.

2.4.2. Protocolos de comunicación (serie)

La comunicación serie se caracteriza por el envío secuencial de información, de forma que transmita un único bit al mismo tiempo por el canal de comunicaciones. Dentro de este tipo de comunicaciones, se debe distinguir entre dos variantes:

Comunicación serie síncrona

Los protocolos serie síncronos cuentan con una señal de reloj que sincroniza la operación de los distintos emisores/receptores. Esta situación se produce en protocolos como *SPI* o *I2C*.

Inter-Integrated Circuit (*I2C*) es un protocolo de transmisión serie síncrono usado principalmente para la comunicación de componentes en un circuito. Está diseñado siguiendo el esquema maestro-esclavo, por lo que el maestro es el encargado de iniciar las transferencias de datos y generar la señal de reloj.

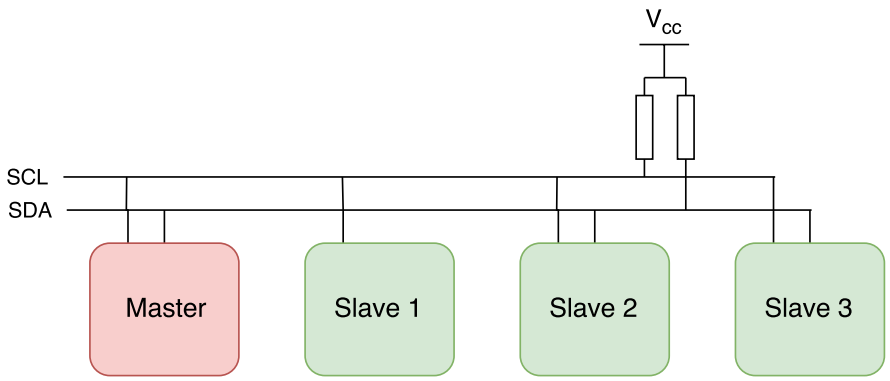


Figura 2.2: Esquema I2C básico con un maestro y tres esclavos

Es posible la existencia de múltiples maestros en el mismo bus. En este caso, los maestros turnarán su operación, de forma que no transmitan al mismo tiempo (pudiendo comunicarse entre ellos también).

Para controlar la transmisión de datos, el estándar cuenta con dos líneas de señal: SCL (Serial Clock), para la transmisión de los pulsos de reloj, y SDA (Serial Data), para la transmisión de los datos (tal como muestra la figura 2.2). Estas dos líneas, cuando el sistema

se encuentra en reposo, están a nivel alto (conectadas a resistencias de pull-up).

La información se envía en mensajes, teniendo cada uno de ellos la forma ilustrada en la figura 2.3.

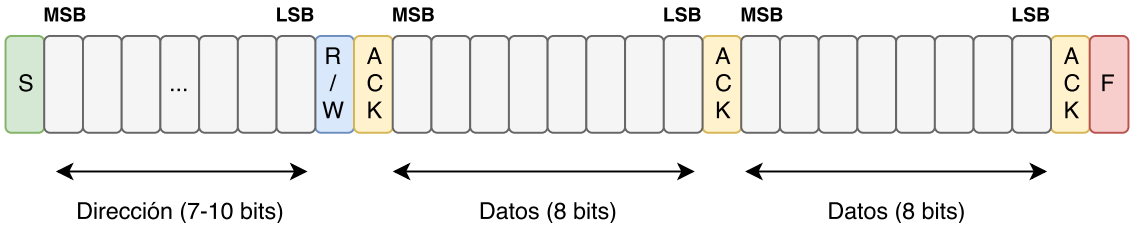


Figura 2.3: Mensaje I2C

Por tanto, el mensaje consta de los siguientes elementos:

- Start: señal de comienzo de la transmisión. La señal de SDA cambia de nivel alto a bajo y posteriormente se produce un cambio equivalente en SCL.
- Address: se indica la dirección que identifica el esclavo con el que se desea interactuar. Esta dirección tendrá una longitud de entre 7 y 10 bits.
- R/W: indica si se quiere realizar una lectura (nivel alto de voltaje) o una escritura (nivel bajo de voltaje)
- ACK: bit para la confirmación de la recepción de los mensajes.
- Data: información a transmitir, que será enviada segmentada en bytes (cada uno de los cuales tendrá asociado un bit de ACK para asegurar una correcta transmisión).
- Stop: señal de parada, consistente en un cambio de bajo a alto nivel en la línea SCL, seguido de un cambio equivalente en la línea SDA (quedando restaurado el estado de reposo, con ambas líneas a nivel alto).

Comunicación serie asíncrona

En este tipo de protocolos no existe una sincronización continua de los módulos que intervienen en la comunicación. En su lugar, se producen señales de comienzo y parada

de la comunicación antes de transmitir cada conjunto de datos. Mientras la transmisión está activa, los emisores/receptores operan siguiendo una frecuencia preestablecida. Esta frecuencia es usualmente establecida en baudios, unidad que especifica el número de símbolos (bits, en el caso de protocolos serie) transmitidos por segundo.

Por otra parte, para detectar la presencia de errores en el canal de comunicación, es común la adición de un bit de paridad tras cada conjunto de datos, de forma que esté activo si hay un número par (paridad par) o impar (paridad impar) de unos en el mismo.

Por tanto, al detallar una comunicación serie se suelen definir los siguientes parámetros:

- Número de bits de comienzo de la transmisión.
- Número de bits de parada.
- Bits por segundo (baudios).
- Número de bits transmitidos en cada secuencia (entre cada par de señales de comienzo y parada).
- (Opcional) Presencia del bit de paridad.

Capítulo 3

Metodología

En este capítulo se detallan los principales aspectos relativos al desarrollo e implementación del sistema de predicción de migrañas. Para comenzar, se comentan los procesos de obtención y formateado de datos y se explica el proceso de generación de los modelos (tanto para predicción como para regeneración de señales). Posteriormente se presenta el entorno simulado en *DEVS*[17], que proporciona un medio para plasmar y validar el sistema y ha servido de referencia para la implementación del predictor de migrañas presentado en este proyecto. Tras ello, se detallan los sensores y mecanismos de entrada utilizados para proporcionar la información sobre las variables biométricas de las que depende el sistema. Una vez definida la entrada, se procede a mostrar la implementación en *VHDL* del sistema (proporcionando detalles sobre su comportamiento, sincronización y optimización). En este punto, teniendo especificada la entrada y el sistema de predicción, se muestran los detalles de la placa desarrollada como interfaz para la interacción con la *FPGA*, que acondiciona y procesa las señales de entrada.

3.1. Definición de modelos predictivos

Esta sección detalla el proceso de obtención de los datos médicos usados en el proyecto. Además, explica el proceso de generación de modelos en base a estos datos, tanto para la recuperación de señales como para la predicción de episodios de dolor.

Este proceso de entrenamiento de modelos se realiza de forma offline y nutre con los

modelos producidos al sistema detallado en este proyecto (que servirá además para validarlos en un entorno real).

3.1.1. Obtención de los datos

Los datos han sido obtenidos mediante la monitorización de pacientes en condiciones ambulatorias, usando dispositivos inalámbricos para el registro de variables biométricas[18]. En concreto, se han usado el PLUX-Wireless Biosignals[20], para la recogida de los valores de sudoración, temperatura y electrocardiograma (ECG), y Nonin Onyx II[13], para la recogida de datos sobre saturación de oxígeno. Los valores de ritmo cardíaco se extraen posteriormente a partir del ECG.

Estos dispositivos envían los datos de los sensores a través de una conexión *Bluetooth* a los móviles usados para el estudio, donde los pacientes marcan la información subjetiva de dolor a medida que sufren los episodios de migraña. Esta información, una vez agrupada, se envía a los servidores oportunos para su posterior procesado.

Figura 3.1: Paciente usando el kit de monitorización

3.1.2. Generación de los modelos

Modelos N4SID

Estos modelos sirven para predecir episodios de dolor. Dado que los datos de dolor introducidos por el paciente vienen representados por una serie de valores discretos en el

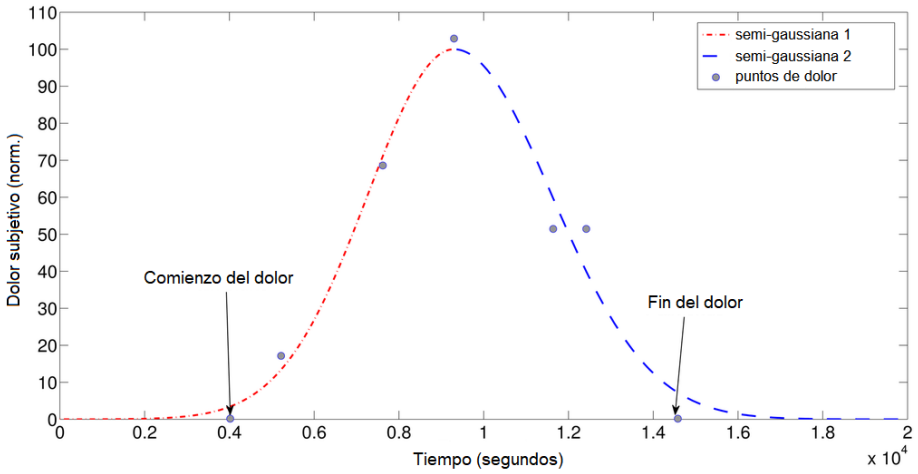


Figura 3.2: Modelado de la curva de evolución de dolor

tiempo, es conveniente pasar esta información a un espacio continuo. Para ello, se localiza el máximo nivel de dolor marcado por el paciente y se realizan dos semi-gaussianas. Una de ellas corresponderá al incremento inicial de dolor, desde el inicio hasta el máximo, y la otra a la progresión y descenso del mismo, desde el máximo al final[18]. Este ajuste aparece representado en la figura 3.2.

Los modelos N4SID se generan en base a los datos de entrada y los niveles de dolor subjetivo marcados por el paciente. Para ello se ha usado System Identification Toolbox, perteneciente al software *Matlab*[12].

Dado que se parte de la hipótesis de que las características del sistema nervioso autónomo (que controla las variables biométricas implicadas en este proyecto) es dependiente de cada paciente, estos modelos son personalizados y deben ser generados para cada uno de ellos.

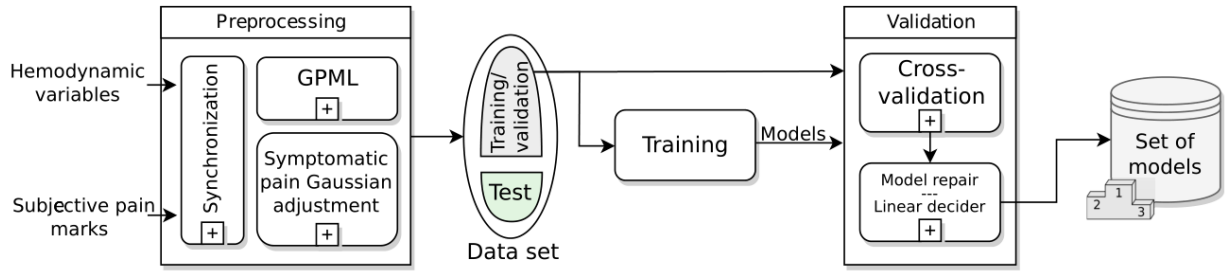


Figura 3.3: Flujo de procesamiento para la generación de modelos

El proceso de generación de estos modelos puede observarse en la figura 3.3 y consta de las siguientes etapas:

- **Preprocesado**: dado que en la recogida de señales se pueden producir pérdidas (por desconexiones Bluetooth o descarga de la batería) es necesario una fase inicial que formatee e intente recuperar los datos perdidos. Para ello, inicialmente se agrupan las distintas variables en intervalos de un minuto mediante un filtro *FIR* (Finite Impulse Response)[10]. Posteriormente, mediante un proceso *GPML* (Gaussian process machine learning) se recuperan los posibles datos perdidos. La implementación de *GPML* usada ha sido desarrollada por Rasmussen[23][21] bajo licencia *FreeBSDLicense*.

- **Generación de los conjuntos de datos**: tras haber preprocesado la señal, los datos resultantes se dividen en dos grupos: uno de entrenamiento (formado por el 75 % de los datos) y otro de validación (correspondiente al otro 25 % de los datos). Los datos asignados para cada uno de estos dos grupos son seleccionados aleatoriamente a partir del conjunto original.

- **Entrenamiento**: fase ejecutada una vez por paciente. Dado que se deben seleccionar varios parámetros para guiar el proceso N4SID, como el número de entradas pasadas o el orden de las matrices (ver ecuación 2.1), se generan diversos modelos usando una optimización por búsqueda exhaustiva (ya que el espacio de búsqueda es limitado). Además, se tienen en cuenta diferentes horizontes de predicción, de forma que se entrenen modelos especializados para detectar episodios con cierto tiempo de antelación (por ejemplo: 10, 20 o 30 minutos).

- **Validación**: fase en la que se obtienen los mejores modelos. Para ello se producen distintas curvas de dolor con cada uno de ellos y se comprueba cuales de ellas tienen un mejor ajuste con la curva generada con los datos proporcionados por los pacientes.

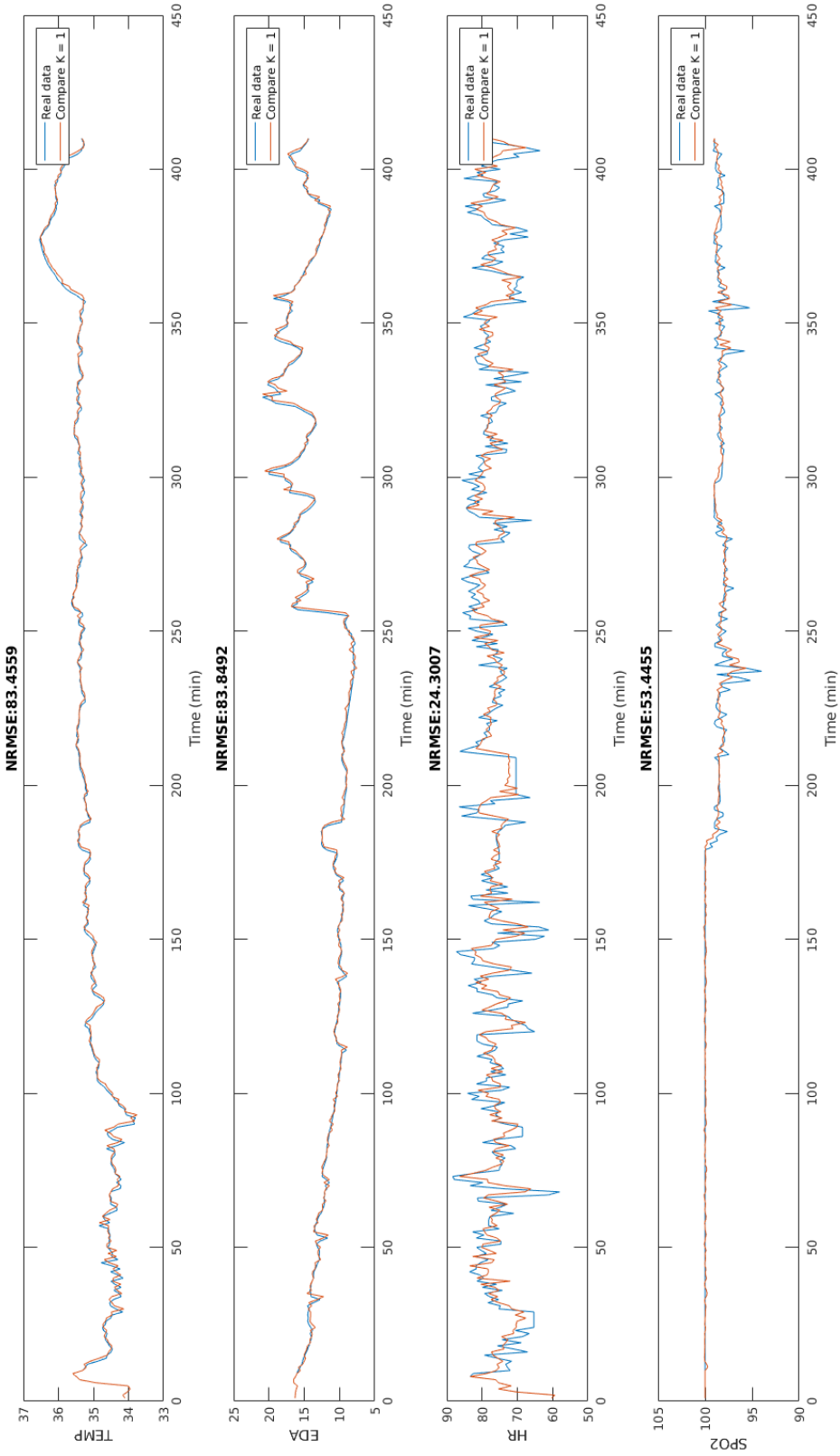


Figura 3.4: Entrenamiento de los modelos ARX

Modelos ARX

Estos modelos se usan para la regeneración de valores de entrada ante la aparición de fallas en la señal.

Para la generación de los modelos ARX toma especial importancia la elección del orden. Este parámetro especifica el número de muestras pasadas a tener en cuenta para el cálculo de los modelos y varía la efectividad de los mismos.

En el proyecto se ha usado un script de *Matlab* que comprueba la efectividad de los órdenes de 1 hasta 10. Para cada orden usado se entrena un modelo con datos del conjunto de entrenamiento (generado de forma similar a como se comentó en el apartado anterior). Posteriormente se comprueba la efectividad de cada uno de ellos usando un conjunto de validación, eligiendo el que presente un mejor resultado (siendo el modelo usado en el proyecto de orden 10).

3.2. Modelo *DEVS*

Para el desarrollo del sistema de predicción se parte de un sistema *DEVS* (representado en la figura 3.5), que simula y valida la plataforma antes de realizar la implementación final[17].

Este sistema simula la entrada de los sensores mediante la lectura e inclusión en la plataforma de datos almacenados en archivos externos. De esta forma, la información sobre las variables a tratar (temperatura, sudoración, ritmo cardíaco y saturación de oxígeno) ya se encuentra formateada y se puede abstraer el uso de los sensores.

Esta información pasa a través de los *Drivers*, que se encargan de añadir una marca de tiempo a las mismas, para su posterior sincronización.

Tras ello, se enrutan los datos a los Sensor Status Detectors (*SSDs*). En ellos, se encuentran los distintos componentes de detección de errores (saturación, caídas o ruido).

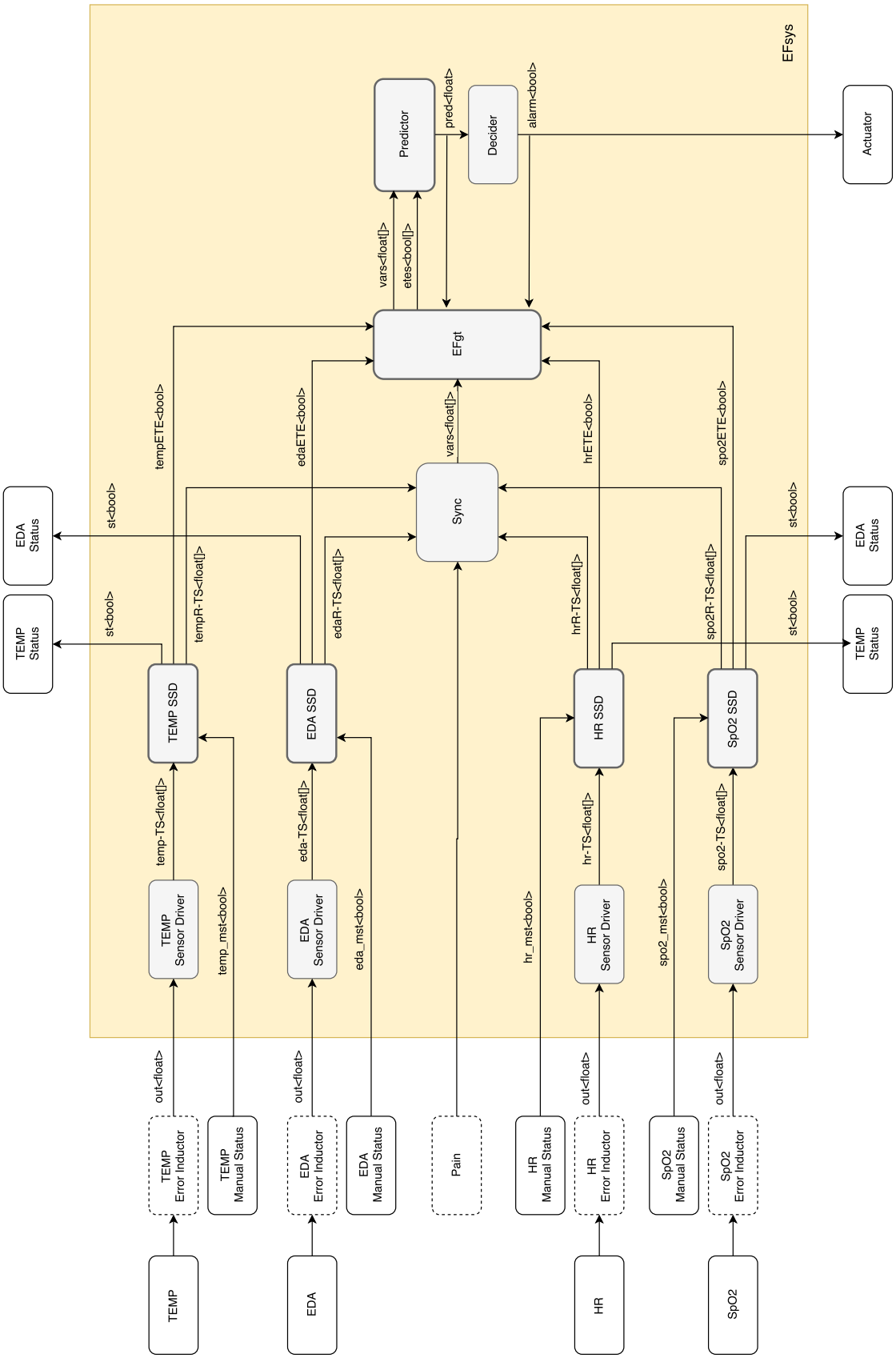


Figura 3.5: Sistema DEVS para la representación del sistema de predicción

Cuando se detecte alguna de estas fallas se activa el componente de regeneración de la señal mediante aprendizaje máquina basado en procesos gaussianos (Gauss Process Machine Learning, *GPML*). Este módulo tiene la función, por tanto, de generar valores válidos de las distintas variables cuando la entrada está sufriendo algún problema de los anteriormente comentados. Para ello, se tienen en cuenta los valores pasados de la variable a reconstruir, de forma que los valores generados respeten la tendencia que tenía la señal de forma previa al fallo. Este mecanismo se usa para contrarrestar fallos temporales de corta duración. En el momento en el que el sistema no pueda seguir generando valores de forma fiable se activan señales que avisan al predictor de que la variable implicada no puede seguir siendo usada (de forma que se tengan en cuenta únicamente las demás variables).

Posteriormente, la información de las distintas variables se agrupa en el sincronizador. Este componente se encarga de unificar las cuatro entradas en base a las marcas de tiempo incluidas junto a las mismas (en los *Drivers*).

Una vez se dispone de paquetes unificados, éstos se enlazan con el *Predictor* (pasando por el EFgt, encargado de generar estadísticas sobre el funcionamiento del sistema). En él, dependiendo del estado de los sensores se escogen los modelos oportunos y se combinan las predicciones resultantes (en el *LinearCombiner*), de forma que se produzca una única predicción como salida en el *Predictor*. Es importante destacar que, aunque en este proyecto se usan modelos de espacio de estados para la generación de predicciones, el diseño permite la introducción de cualquier otro tipo de modelo adecuado para la generación de este tipo de información (pudiendo incluso combinarlos si se cree oportuno).

El sistema cuenta con cinco salidas: cuatro indicadores (uno por variable) que indican la caída de uno de los sensores y un actuador que alerta de la proximidad de un episodio de migraña.

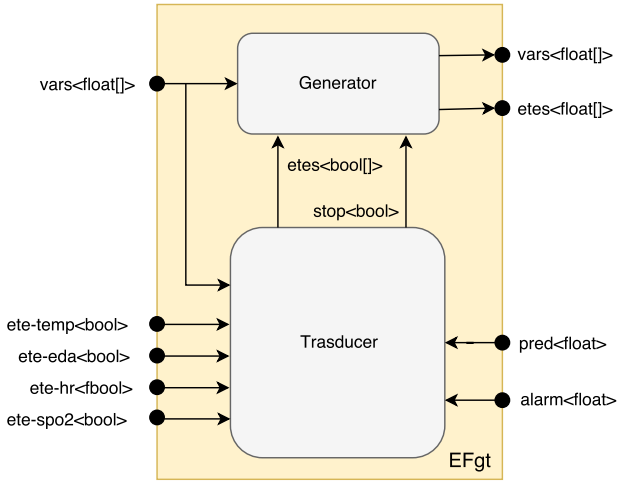
En los siguientes apartados se describen los distintos componentes del sistema, comen-

zando por los componentes acoplados (formados por otros componentes) y terminando con los atómicos.

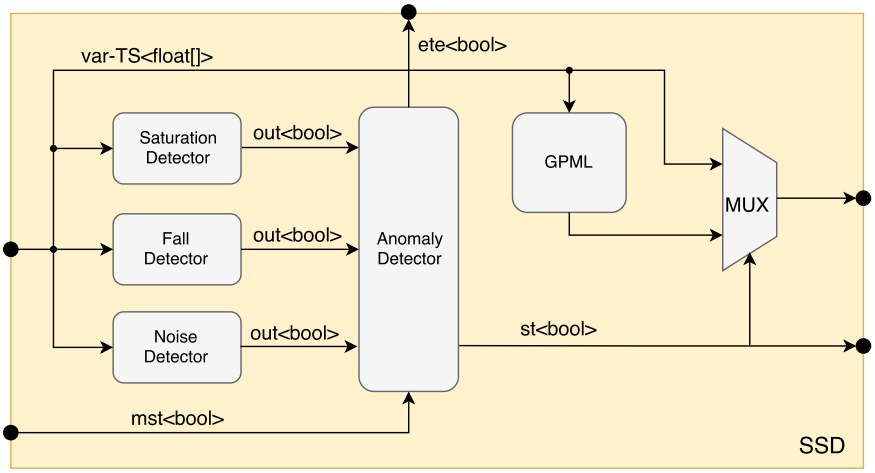
3.2.1. Componentes acoplados

Este sistema cuenta con los siguientes componentes acoplados (diferenciados en la figura 3.5 con un borde de mayor grosor):

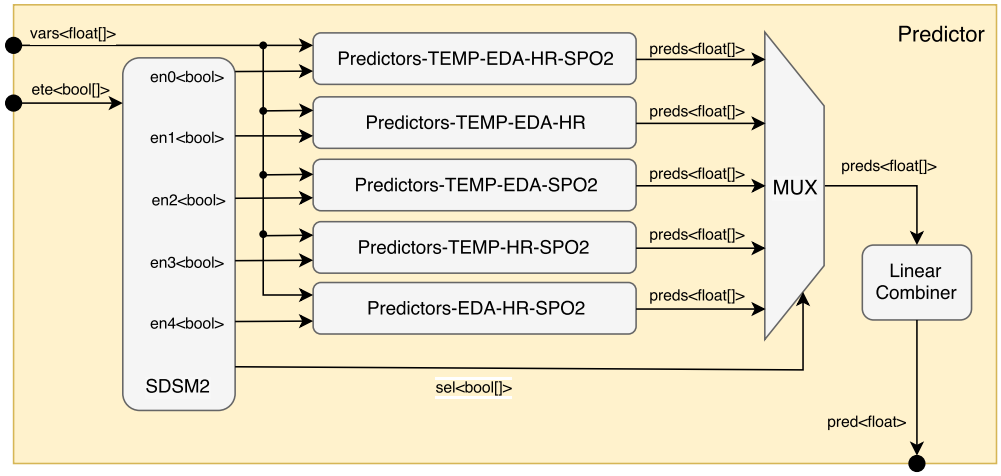
- *EFsys*: representa el sistema de predicción de migrañas completo. Recibe las entradas de los sensores y los pulsadores de restablecimiento de los mismos y genera una señal para activar, cuando sea oportuno, la alerta de aparición de dolor. Además, mantiene cuatro salidas adicionales para informar de fallos en los distintos sensores.
- *SSD* (figura 3.6b): acrónimo de Sensor Status Detector. Contiene tres componentes de detección de errores: *SaturationDetector* para las detectar saturaciones, *FallDetector* para detectar caídas y *NoiseDetector* para detectar la presencia de ruido en la señal entrante. Para los dos primeros, se activará la señal de error cuando el promedio de los últimos valores de la señal sea mayor o menor que límites preestablecidos (respectivamente). Para la detección de ruido, se activará la salida cuando el promedio de los cuadrados de los últimos valores sea mayor que un umbral (entrenado anteriormente con varias horas de datos). Las salidas de estos tres componentes se enlazan a un detector de anomalías (*AnomalyDetector*), encargado de producir dos salidas: una que informa cuando hay un fallo en el sensor (y se activará cuando uno de los tres detectores notifique una falla en la señal) y otra produce una alerta en el momento en el que no se puede seguir regenerando los valores del sensor caído de forma fiable (*Elapsed Time Exceeded*, *ETE*). Por tanto, la señal *ETE* señala el momento en el que la variable implicada debe dejar de ser usada para la generación de predicciones. Por otra parte, la reconstrucción de los valores de las variables de entrada se realizará cuando el



(a) EFgt: genera estadísticas con los datos del sistema



(b) SSD: comprueba el estado de los sensores



(c) Predictor: genera predicciones

Figura 3.6: Componentes acoplados del *EFsys*

detector de anomalías reporte un fallo presente en la señal, en el módulo *GMPL* (en base a los valores pasados de la variable implicada).

- *EFgt* (figura 3.6a): compuesto por dos componentes atómicos: un generador (*Generator*) y un transductor (*Transducer*). Los paquetes generados por el sincronizador pasan a través del generador y, junto a las predicciones resultantes y la información de posibles errores continuados (ETEs) en las señales, producen estadísticas de funcionamiento del sistema en el transductor.

- *Predictor* (figura 3.6c): recibe la mediciones de los sensores y la información sobre su estado y genera una predicción indicando la posibilidad de ocurrencia de un nuevo episodio. Para ello, mantiene varios modelos de espacio de estados (*N4SID*) previamente entrenados para la predicción de episodios de dolor. Además, como los sensores pueden dejar de producir datos válidos, se manejan varios modelos para lidiar con cada conjunto de variables (de al menos tres elementos). En concreto, se mantienen tres modelos para cada conjunto de variables posible (agrupados en componentes *ModelSet*). Por tanto, se dispone de cinco conjuntos de modelos, de forma que se pueda predecir el grado de dolor usando todas la variables del sistema o cualquier subconjunto en el que se omita una de ellas. De esta forma, no es posible esta operación cuando se están produciendo fallos en dos de las entradas simultáneamente (ya que la salida proporcionada no sería suficientemente fiable teniendo en cuenta dos o menos variables). El conjunto de modelos activo en cada momento es determinado por el componente atómico Sensor Dependant Model Selection System (*SDMS2*), en base a las señales *ETE* de los distintos *SSD* (que se activan cuando los errores en una variable se han mantenido el tiempo suficiente como para que no sea fiable tenerla en cuenta en los mecanismos de predicción). Las tres predicciones producidas se combinan (haciendo un promedio) en el componente *LinearCombiner*, de forma que el *Predictor* genere una única predicción.

3.2.2. Componentes atómicos

Este sistema cuenta con los siguientes componentes atómicos:

- *TEMP*, *EDA*, *HR*, *SPO2*: representan las cuatro variables biométricas contempladas en el proyecto (temperatura, sudoración, ritmo cardíaco y saturación de oxígeno). Cada una de ellas representa la entrada de un sensor, cuyos datos se almacenan en archivos externos.
- *[TEMP | EDA | HR | SPO2] Error Inductor*: módulos que pretenden simular los errores comentados anteriormente. (saturación, desconexiones y ruido eléctrico o distorsión en la señal).
- *[TEMP | EDA | HR | SPO2] Manual Status*: pulsadores que permiten indicar el restablecimiento de uno de los sensores cuya actividad se haya visto interrumpida previamente. En un entorno real serían usados por el paciente tras recolocar o cambiar un sensor tras la aparición de un fallo.
- *[TEMP | EDA | HR | SPO2] Sensor Driver*: añaden una marca de tiempo a las variables entrantes. En caso de que los datos no estuvieran previamente convertidos a las magnitudes físicas correspondientes estos componentes se encargarían de realizar esta conversión antes de proporcionárselos al siguiente componente. Además, si se tratará de una implementación en un entorno real también asumirían la gestión de los mecanismos de entrada (teniendo que gestionar los protocolos de comunicación oportunos). Además el driver correspondiente al ritmo cardíaco extraerá su información de los datos de electrocardiograma (*ECG*) entrantes.
- Sincronizador (*Sync*): se encarga de almacenar y sincronizando las distintas variables para crear paquetes unificados que representen los valores de los sensores en un mismo instante de tiempo. Para ello, se usa un diezmado con un filtro *FIR*. Este componente se hace necesario debido a la posible distinta frecuencia de operación de cada uno de los sensores pertenecientes al sistema.

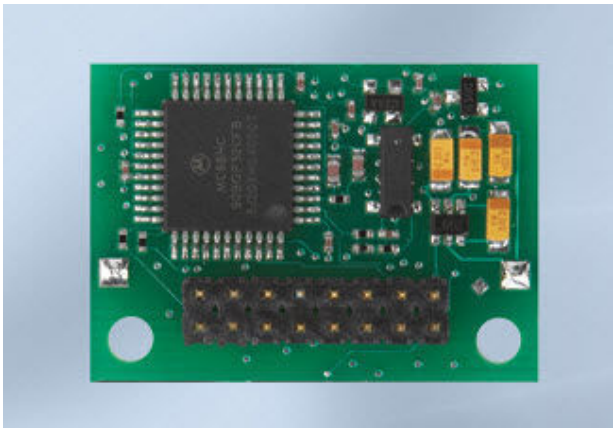


Figura 3.7: Transductor para el sensor de pulsioximetría (OEM III)

- *Decisor*: determina si se va a producir un nuevo episodio de dolor en base a la predicción que le llega como entrada. Considerará que se va a producir un evento cuando se supere un umbral.

3.3. Incorporación de E/S

En esta sección se describen los distintos dispositivos usados para la recogida de las variables biomédicas. Para ello, se divide la descripción en dos apartados: uno que detalla los sensores con salida digital y otro para los que generan salida analógica (con los que se usará un conversor A/D para adaptar la señal).

Los sensores mostrados en este apartado (y, por tanto, usados en el sistema final) no se corresponden son distintos a los usados por la recogida de datos en fases previas. Además, en vez de recoger información de *ECG* se usa la información de ritmo cardíaco proporcionada por el sensor de pulsioximetría (cuyo ASIC es el mismo que el del *Onyx*).

3.3.1. Entrada digital

Para la recogida de las medidas de saturación de O_2 (SpO2) y ritmo cardíaco (HR) se ha usado el dispositivo Nonin OEM III[14] (ver figura 3.7). Este sensor comunica los

datos con una salida serie con las siguientes características:

- 9600 bits por segundo.
- 8 bits de datos.
- 1 bit de parada.
- Sin paridad.
- Sin control de flujo.

Por otra parte, cuenta con tres formatos para la comunicación de las mediciones recogidas, diferenciándose por el nivel de detalle y la frecuencia con la que se informa de nuevos valores:

- Formato 1: manda una muestra por segundo (3 bytes), con información de ritmo cardíaco y saturación de oxígeno.
- Formatos 2 y 7: manda tres paquetes por segundo. Cada paquete contiene información detallada de SpO2 y HR.

Todos estos formatos cuentan además con información relativa al estado de la señal, pudiendo obtener información sobre desconexión del sensor y calidad de los datos leídos.

La selección del modo de operación se realiza mediante el cambio de una resistencia, definiéndose tres rangos que se corresponden con los tres formatos citados. Además, todos ellos contienen información sobre la calidad de la señal, permitiendo conocer si el sensor está desconectado o proporciona malas mediciones.

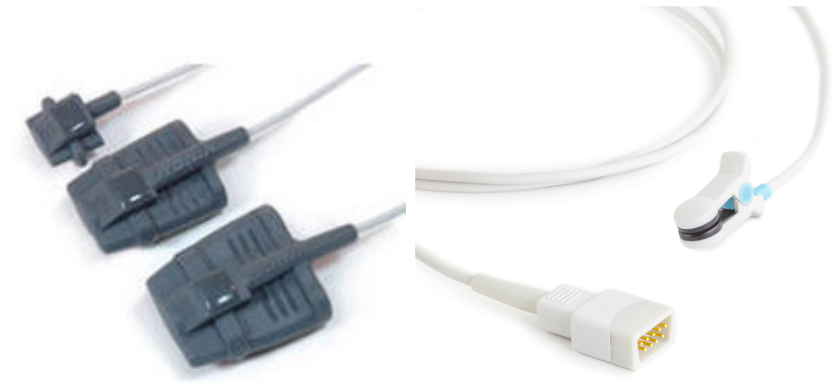
El formato 1, usado para este proyecto, tiene la estructura mostrada la figura 3.8. Dispone, por cada muestra, de 5 bits para informar sobre el estado de la señal, 9 para codificar las mediciones de ritmo cardíaco, 7 para las de SpO2 y 3 bits fijos (de control). Por otra parte, se disponen de sensores compatibles para el dedo y para la

Byte 1 - Status							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
1	SNSD	OOT	LPRF	MPRF	ARTF	HR8	HR7
***Note**: Bit 7 is always set							

Byte 2 - Heart Rate							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	HR6	HR5	HR4	HR3	HR2	HR1	HR0
***Note**: Bit 7 is always clear							

Byte 3 - SpO2							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	SP6	SP5	SP4	SP3	SP2	SP1	SP0
***Note:** Bit 7 is always clear							

Figura 3.8: Formato de salida disponibles en OEM III (1)



(a) Para el dedo (b) Para la oreja

Figura 3.9: Sensores de fotopletismografía

oreja (figura 3.9). En el transcurso del proyecto se ha usado principalmente el primero de ellos.

3.3.2. Entrada analógica

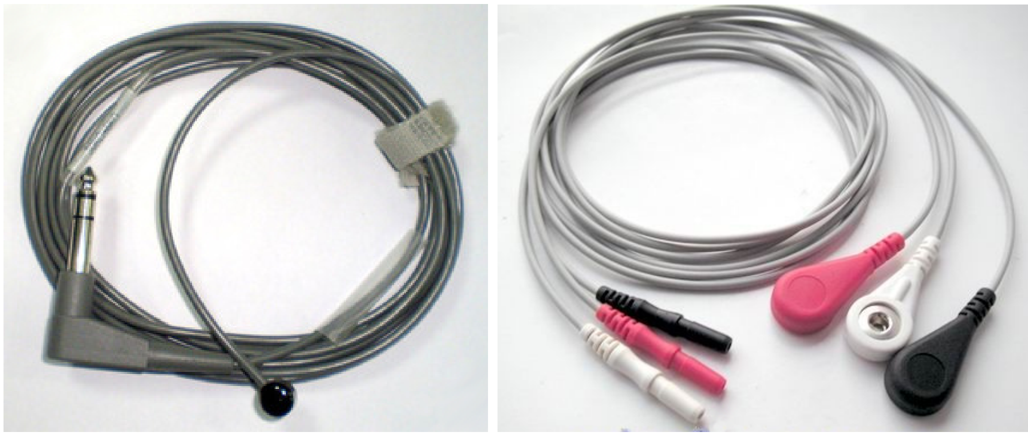
Para la obtención de las medidas de temperatura y sudoración se han usado sensores analógicos como los que se pueden ver en la figura 3.10. En la figura 3.10a se muestra un termistor *NTC* preparado para tomar mediciones en cuerpos humanos (sensor de uso clínico). En la figura 3.10b se muestran los sensores usados para tomar medidas de sudoración (se usan dos de estos sensores).

Para tratar con estos datos, se convierten a digital antes de ser usados por el sistema mediante un conversor analógico/digital (A/D). Para asegurar la compatibilidad con posibles cambios de FPGAs se ha optado por usar un conversor A/D externo, el PModAD2[5]. Este dispositivo tiene las siguientes características:

- Doce bits de resolución.
- Cuatro canales de conversión.
- Comunicación por I2C.
- Voltaje de referencia de 2.048 V.
- Posibilidad de usar alimentación de referencia externa.

Por tanto, la *FPGA* debe disponer de un componente que interprete el protocolo *I2C*. Este componente actúa como maestro y debe configurar los parámetros de medición de forma previa a la lectura de datos. Para ello, se debe escribir en la dirección correspondiente al dispositivo un byte con el formato especificado en la figura 3.12, donde:

- CH3 - CH0: permiten la activación de los canales de entrada (1: activado, 0: desactivado). En el caso de seleccionar más de un canal simultáneamente se envían las mediciones de los canales elegidos en secuencia, empezando desde el canal



(a) Sensor de temperatura (b) Sensor de sudoración

Figura 3.10: Sensores con salida analógica

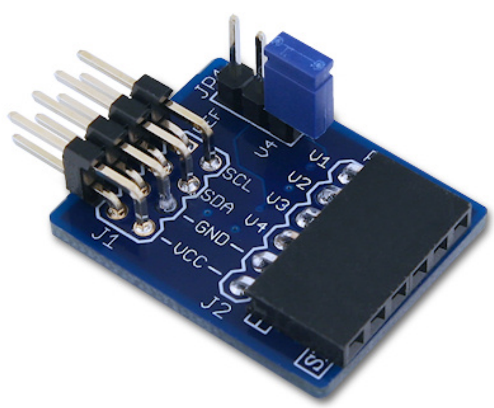


Figura 3.11: Conversor A/D: PmodAD2

D7	D6	D5	D4	D3	D2	D1	D0
CH3	CH2	CH1	CH0	REF_SEL	FLTR	Bit trial delay	Sample delay
1	1	1	1	0	0	0	0

Figura 3.12: Byte de configuración (PModAD2), con sus valores por defecto

más bajo. Para cada uno de los canales se envian dos bytes (de más a menos significativo), siendo relleno los cuatros primeros bits de cada muestra. Por defecto están activados los cuatro canales. En este proyecto se activan únicamente los dos primeros, en los que se recogen los datos de temperatura y sudoración (respectivamente).

- REF_SEL: permite establecer una fuente externa como referencia. Si se establece a 0, se usa la alimentación de la placa como referencia. Si se establece a 1 se usa una fuente externa que deberá suministrarse en la entrada reservada para el canal 3 (restringiendo la operación del dispositivo a un máximo de 3 canales).
- FLTR: activa un filtro en las líneas SDA y SCL para la reducción de ruidos (1: activado, 0: desactivado).
- Bit Trial Delay / Sample Delay: permiten introducir retrasos en ciertas operaciones del conversor para evitar que coincidan con actividad en el módulo I2C.

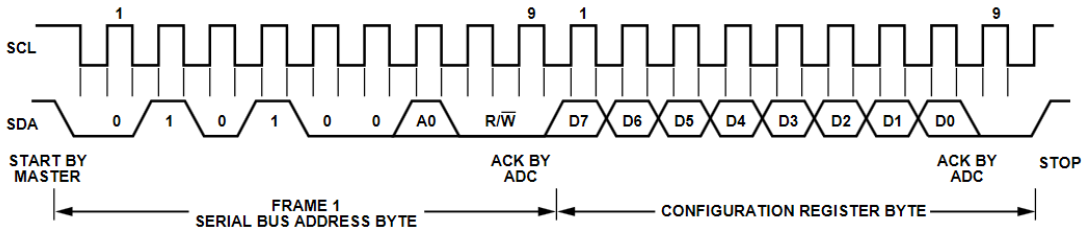


Figura 3.13: Configuración del módulo PModAD2 mediante I2C

En la figura 3.13 se puede observar el proceso mediante el cual se configura el conversor A/D: inicialmente se especifica la dirección del dispositivo y se mantiene a nivel bajo el bit R/W para escribir en el dispositivo. Posteriormente se envía el byte correspondiente a la configuración elegida.

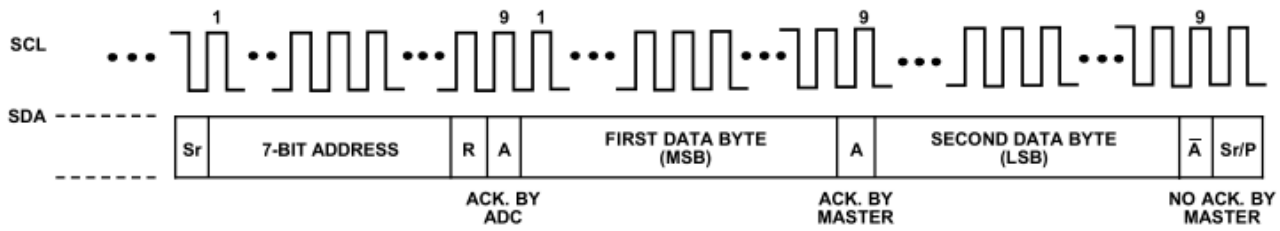


Figura 3.14: Lectura de la salida digital del PModAD2

Tras la configuración, se puede leer el dispositivo. Para ello, se escribe la misma dirección manteniendo a nivel alto el bit de R/W (indicando que se quiere realizar una lectura). Este proceso aparece representado en la figura 3.14. En el proyecto, dado que se usan dos canales, se devuelven cuatro bytes por lectura (dos para cada variable).

3.4. Diseño FPGA

En este apartado se comentan las herramientas, entornos y recursos utilizados para la implementación del sistema de predicción de migrañas en una *FPGA*. Para ello se usa *VHDL* como lenguaje de especificación hardware y Xilinx ISE como entorno de verificación[27] (usando la utilidad Impact, de la misma compañía, para la comunicación con la placa y la subida de archivos).

Durante la implementación se toma como referencia el sistema *DEVS* detallado en el apartado 3.2.

En primer lugar se introducen las características de la placa de desarrollo escogida para la realización del proyecto. Posteriormente se detallan los cambios en la arquitectura y comportamiento del sistema introducidos para la adaptación del sistema simulado en *DEVS* a la implementación en *VHDL*. Por último, se listan las señales que permiten la sincronización entre los distintos componentes del sistema.

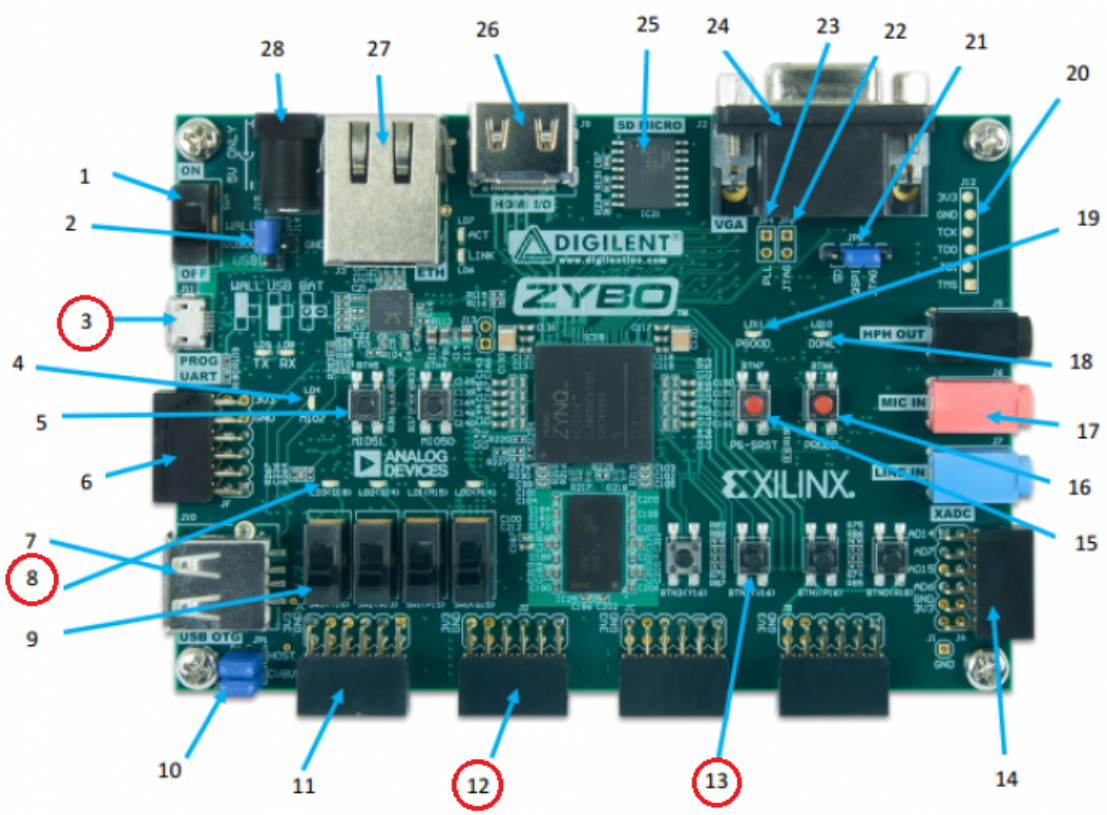


Figura 3.15: Placa de desarrollo: Zynq-7000

Cuadro 3.1: Componentes presentes en la placa de desarrollo Zynq-7000

Id.	Descripción	Id.	Descripción
1	Interruptor de encendido	15	Reset (procesador)
2	Puente (selección de voltaje)	16	Reset (configuración)
3	Puerto USB (UART/JTAG)	17	Conectores de audio
4	LED MIO	18	LED (configuración finalizada)
5	Pulsadores MIO (2)	19	LED (alimentación)
6	Pmod MIO	20	Puerto JTAG
7	Conectores USB OTG	21	Puente (modo de programación)
8	LEDs (4)	22	Puente (habilitación JTAG)
9	Interruptores (4)	23	Puente PLL
10	Puente (selección modo USB)	24	Conector VGA
11	Pmod estándar	25	Conector MicroSD
12	Pmods de alta velocidad(3)	26	Conector HDMI
13	Pulsadores (4)	27	Conector Ethernet (RJ45)
14	Pmod XADC	28	Jack de alimentación

3.4.1. Placa de desarrollo

Durante el desarrollo del proyecto se ha usado la placa de desarrollo Zynq-7000[6]. En la figura 3.15 aparecen señalados los principales componentes de dicha placa, estando detallados en la tabla 3.1. La Zynq-7000 integra un procesador ARM Cortex-A9 con una FPGA Artix-7 de Xilinx (usada en este proyecto). Las características principales de esta *FPGA* son las siguientes:

- 4400 unidades lógicas, cada una con cuatro LUT de 6 entradas y 8 flip-flops.
- 240 KB de Block RAM.
- 80 DSPs (Digital Signal Processors).
- Velocidades de reloj interno superiores a 450MHz.
- Conversor analógico-digital integrado (XADC).

Los identificadores de los recursos de la placa de desarrollo usados en el proyecto aparecen rodeados en la imagen 3.15 y remarcados en la tabla 3.1. Se usan de la siguiente manera:

- Los pulsadores se usan para que indicar el restablecimiento de un sensor caído.
- Los LEDs se usan para indicar el estado de las cuatro variables.
- Las entradas correspondientes a los protocolos serie e *I2C* se establecen en los puertos *Pmod*.
- La salida que indica la proximidad de un nuevo episodio se relacionará con uno de los pines proporcionados por los *Pmod*.

Por otra parte, cabe destacar que la Zynq-7000 no ha sido la primera opción considerada en este proyecto. En las primeras fases del mismo se utilizó la placa de desarrollo XuLA2-LX9[3]. Sin embargo, cuando avanzó el desarrollo del proyecto se determinó que no disponía de suficientes recursos para soportar el sistema de predicción, por lo que

se dio el salto a la placa de desarrollo actual. Las características de la XuLA2-LX9 se detallan en el apéndice B.

3.4.2. Cambios respecto al sistema simulado (*DEVS*)

Para la implementación del sistema en la *FPGA* se usará *VHDL*. El diagrama general (excluyendo señales de sincronización) se puede ver en la figura 3.16. En él se puede observar varios cambios introducidos en esta etapa para adaptar el sistema a esta implementación:

- Se incluyen dos *Drivers*, en vez de los cuatro que se planificaron anteriormente. Esto es debido a que se usa el sensor Nonin OEM III para recoger valores tanto de saturación de oxígeno como de ritmo cardíaco y se leen los valores de temperatura y sudoración a través del mismo conversor A/D (PModAD2), que proporciona los datos en secuencia.
- Se introduce el componente *BuffersHandler*, que lee las salidas del componente *Sync* y almacena un determinado número de muestras pasadas (actualmente las correspondientes a un intervalo de tiempo de diez minutos). Esta información se le proporcionará a los *SSDs*, ya que el componente de recuperación de datos necesita esta información para operar.
- Se elimina el componente EFgt, ya que únicamente tenía el propósito de generar estadísticas sobre los obtenidos por el simulador.
- Se elimina la introducción de las marcas de tiempo junto a los valores de entrada, ya que no son necesarias para la recuperación de valores con el método elegido (*ARX*). Posteriormente el sincronizador hace un promedio de las señales de cada variable cada minuto, por lo que tampoco necesita esta información.
- Para la representación de valores con decimales se usa una codificación en coma fija, tal y como se explica en el apéndice A.

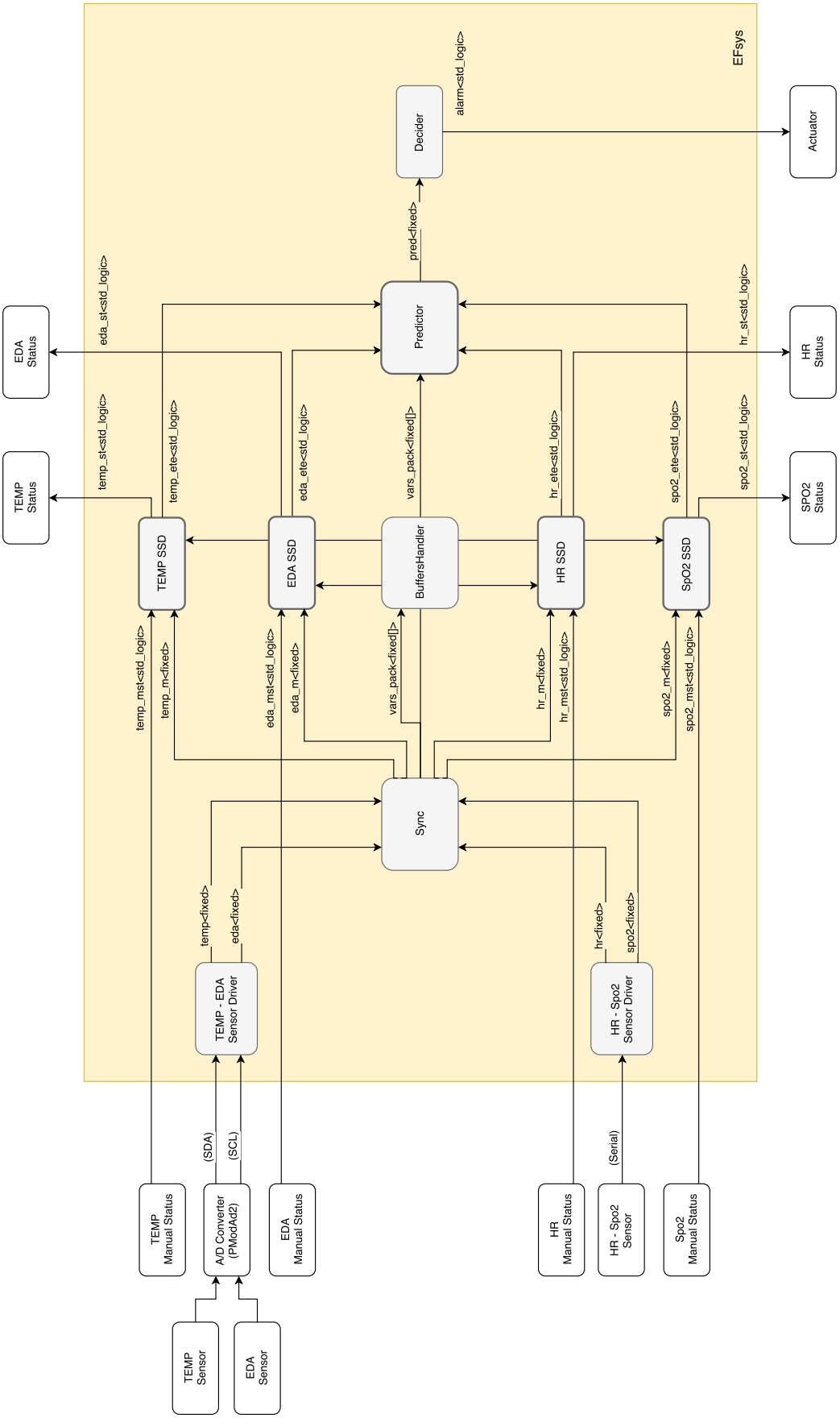


Figura 3.16: Arquitectura del sistema de predicción migrañas en VHDL

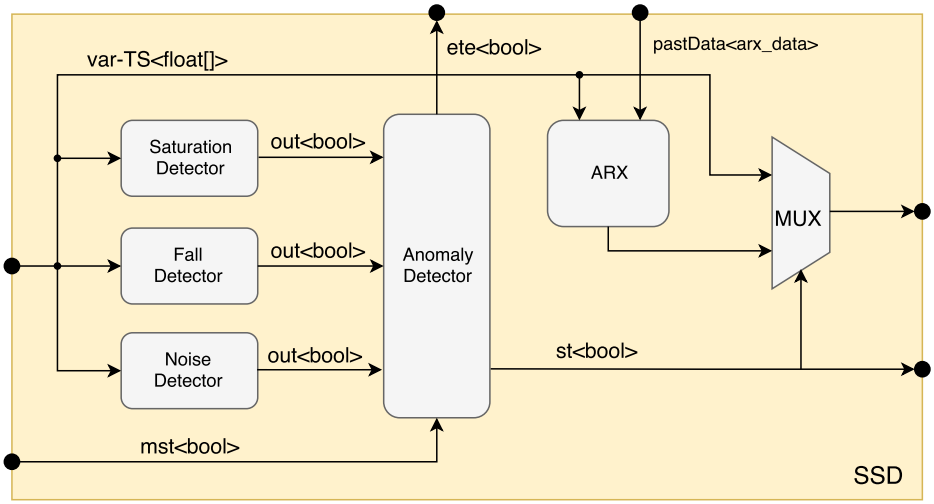


Figura 3.17: Arquitectura módulo de detección de errores (VHDL)

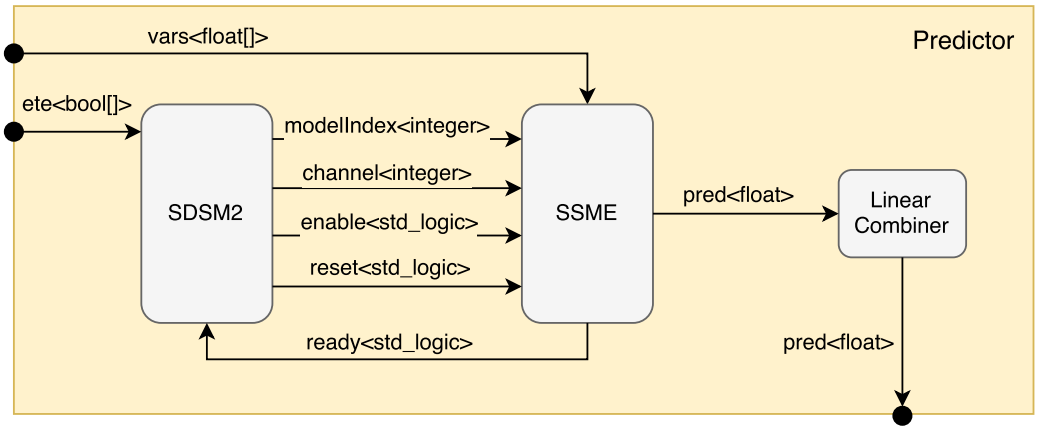


Figura 3.18: Arquitectura módulo de predicción (VHDL)

- Los *SSDs* actúan sobre la salida del sincronizador, en vez de actuar directamente sobre los *Drivers*. En el sistema *DEVS* los ficheros leídos para proporcionar los datos al sistema contenían directamente los datos por minutos, por lo que ese orden era válido. Sin embargo, en la implementación sobre la *FPGA* los datos se agrupan en minutos en el sincronizador. Por lo que los detectores de errores y la recuperación de señales deben actuar sobre la salida de este componente.

Por otra parte, en los componentes de detección de errores (figura 3.17) se ha cambiado el mecanismo de predicción de *GPML* a *ARX*. Este componente necesita las entradas

pasadas tanto de la variable relativa al *SSD* al que pertenece como de las variables exógenas. Por tanto, como todos los *SSDs* deben tener acceso a los valores pasados de todas las variables, se hizo conveniente externalizar este componente para centralizar la información (creando así el componente *BuffersHandler*, comentado anteriormente).

En el predictor (*Predictor*), visible en la figura 3.18, se ha cambiado la arquitectura presente del simulador *DEVS* (donde se instanciaba un componente para cada conjunto de modelos), de forma que la evaluación de la predicción queda unificada en un único componente (*SSME*). Esta versión unificada cuenta con tres canales de información, de forma que se almacenen simultáneamente los tres estados de los que dependen las predicciones que se estén generando. Estas predicciones son calculadas de forma secuencial, controlando el componente *SDMS2* dicho proceso de la siguiente forma:

- Cuando el *SDMS2* recibe un pulso desde el sincronizador (indicando la presencia de un nuevo paquete de datos) inicia una conversión en el *SSME*. Para ello, indicando con la señal *modelIndex* los modelos a escoger (en función de las variables activas en ese momento), se genera un pulso en la señal de *enable* (teniendo seleccionado el canal 0).

- Cuando la primera conversión finalice (hecho comunicado a través de la señal *ready*) se incrementa el número de canal y se genera otro pulso en la señal de *enable* para iniciar la siguiente conversión. Este comportamiento continúa hasta que se hayan generado las tres predicciones, momento en el cual se espera el siguiente pulso del sincronizador. En el capítulo 4 se comprueba que no se producen condiciones de carrera durante la ejecución de este proceso.

- Cuando se alteren las señales *ETE* que llegan al predictor (cuando cambien las variables disponibles para generar la predicción) se produce un cambio en la señal *modelIndex* del *SDMS2* (de forma que se usen los modelos correctos) y se genera un pulso en la señal de *reset* para reiniciar los estados generados en anteriores

predicciones.

3.4.3. Inicialización

Se puede inicializar (o reiniciar) el sistema generando un pulso en la señal *iRst* del componente raíz del sistema de predicción. Cuando se produce este evento se aplica la siguiente secuencia de operaciones:

- Se envía un pulso de *Reset* a todos los componentes síncronos del sistema.
- Tras el pulso, todos reinician su operación y comienzan a actuar. Sin embargo, dado que los módulos de recuperación de señales necesitan diez muestras de minutos pasados, no se generarán alertas de errores hasta que se obtenga esta información.
- Tras ese tiempo el sistema empieza a funcionar por completo, incluyendo la generación de alertas por errores y la recuperación de las señales.

3.4.4. Operación

Tras los cambios previamente comentados, y una vez inicializado el sistema, se sigue el siguiente procedimiento:

- La datos generados por los sensores se enrutan a los *Drivers* correspondientes. Las medidas de saturación de oxígeno y ritmo cardíaco se transmiten por protocolo serie al *HrSpO2Driver* (obteniendo una muestra por segundo) y los valores de temperatura y sudoración se transmiten por I2C, tras pasar por el conversor A/D, hacia el *TempEDADriver* (obteniendo tres muestras por segundo). En ellos se interpretan los protocolos, se calcula la magnitud física correspondiente, se convierten las medidas obtenidas al tipo de dato adecuado (coma fija) y se envían los datos al sincronizador.

- En el sincronizador se recogen todas estas mediciones y se van acumulando en diversos contadores. Cada minuto, se recibe un pulso que provoca que se realice el promedio de los datos obtenidos hasta el momento de cada una de las variables del sistema. Posteriormente se manda como salida un paquete que unifica los resultados. Además, al mismo tiempo, se activa una señal de *Ready* por la salida conectada a los *SSDs*, de forma que estos realicen las operaciones oportunas con los nuevos datos. Después de esto el sincronizador espera un determinado número de ciclos (correspondiente al número de ciclos que los componentes de regeneración de señales tardan en generar un resultado, en caso de ser preciso) y activa una segunda señal de *Ready*, conectada al *Predictor*.

- En los *SSDs*, tras ser notificados de la existencia de un nuevo dato, los módulos detectores de errores guardan el nuevo dato en sus registros de desplazamiento y operan de la manera comentada anteriormente. En caso de haberse notificado alguno de las fallas que detecta el sistema se calculará un valor adecuado para la señal en base a la tendencia de los datos pasados. La salida de cada uno de los cuatro *SSDs* (con el dato original del sincronizador o el valor recuperado correspondiente) se dirigirá al *Predictor*.

- En el *Predictor*, una vez se ha recibido el pulso del sincronizador, se inicia el proceso de generación de predicciones comentado con anterioridad (en el que el *SDMS2* realiza peticiones al *SSME* para generar las tres predicciones oportunas)). Estas predicciones son recogidas y agrupadas por el *LinearCombiner*, de forma que se genere un único dato.

- La predicción generada por el *Predictor* se evalúa, de forma que se active la alarma si supera un determinado umbral[18].

3.4.5. Optimización hardware

Durante la implementación se han realizado varias modificaciones sobre los planteamientos iniciales con la finalidad de reducir los recursos empleados por la *FPGA*. Los principales son los siguientes:

- Para la realización de promedios, en los módulos de detección de errores y en el *Decisor*, en lugar de sumar los distintos datos y dividir posteriormente entre el número de elementos sumados se aplican las comprobaciones directamente sobre la suma. Esto es posible porque el número de elementos es fijo y, por tanto, pueden precalcularse valores adecuados sin necesidad de aplicar divisiones (los registros de desplazamiento en los detectores de errores tienen una longitud preestablecida y el *LinearCombiner* siempre agrupa grupos de tres predicciones.
- Por otra parte, en el sincronizador se debe calcular el promedio de los datos generados cada minuto (de las cuatro variables). Dado que se realizan posteriormente operaciones sobre estos valores (y no es recomendable operar sobre la suma de los mismos, ya que se opera con números de precisión fija y podrían existir problemas de desbordamiento), se multiplica cada uno de los valores por la inversa del número sobre el que se quiere dividir (precalculado). De nuevo, esto es posible porque los *Drivers* han sido configurados para operar a una frecuencia específica (una muestra por segundo para los valores generados por el *HrSpO2Driver* y tres por segundo en el caso de los generados por el *TempEDADriver*).
- En cada uno de los cuatro *ARX* se ha introducido un módulo auxiliar que agrupa las operaciones que se realizan durante la recuperación de señales (multiplicaciones de matrices). De esta forma, se fuerza que el sintetizador use los mismos recursos de la *FPGA* en cada iteración del proceso, reduciendo notablemente los componentes a utilizar.
- Anteriormente, en el sistema de simulación, se disponía de tres evaluadores de

modelos para cada conjunto de variables (agrupados cada uno de ellos en un *ModelSet*). Esto provocaba la existencia de 15 evaluadores independientes en el sistema y se traducía en un consumo elevado de recursos. Por este motivo se agruparon todos estos evaluadores en un único componente, que aglomera las operaciones oportunas y mantiene tres canales para la gestión de los estados relativos a cada una las tres evaluaciones activas en cada momento.

3.4.6. Sincronización de componentes

La correcta sincronización del sistema depende de una serie de señales de reloj, marcas temporales y pulsos (representadas en la figura 3.19):

- Reloj principal (*Main clk*): dependiente de la frecuencia base de la placa de desarrollo. Dado que se usará la Xilinx Zynq Z-7010 esta frecuencia corresponde a 125 MHz. Con este reloj operan los Drivers para comunicarse con los sensores y conversores de entrada.
- Reloj de operaciones (*Op. clk*): obtenido a partir del reloj principal con un divisor de frecuencia (*FreqDiv*). Es la frecuencia de principal del sistema y es utilizada por los módulos de recuperación y predicción (*ARX* y *SSME*) y por el sincronizador. Opera a 100KHz.
- Reloj de temporización: derivado del reloj de operaciones con otro divisor de frecuencia. Opera a 100 Hz.
- Timestamp: marca de tiempo generada con el módulo *TimestampGen* usando como referencia el reloj de temporización. Por tanto la marca de tiempo del sistema ofrece una precisión de una centésima de segundo. Esta señal será usada por los módulos de detección de anomalías (*AnomalyDetector*), dentro de los *SSDs*, para generar la señal ETE (que se activa cuando los errores en un sensor perduran un determinado tiempo preestablecido).

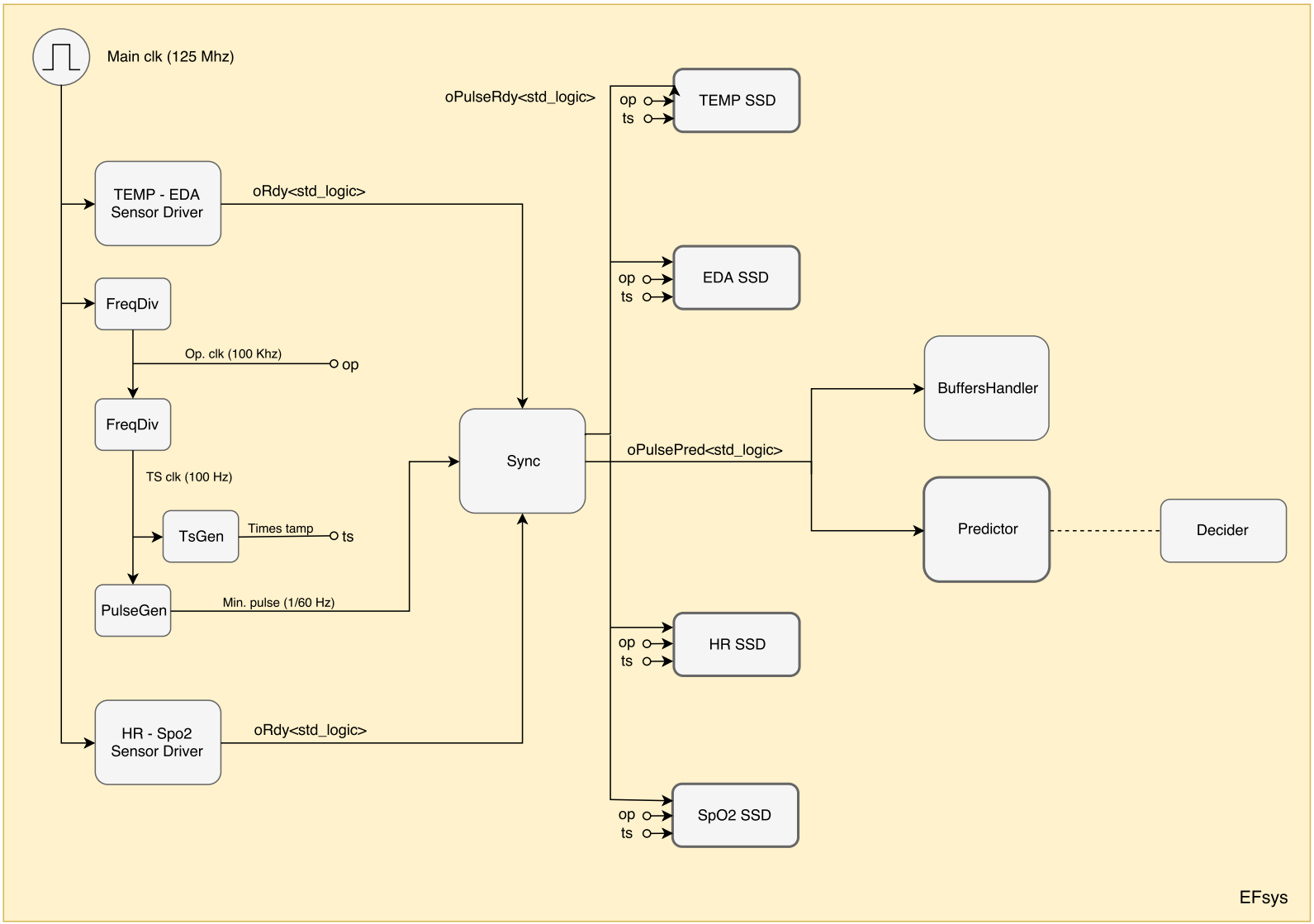


Figura 3.19: Mecanismos de sincronización y temporización del sistema VHDL

- Pulso por minutos (*Min. pulse*): generado a partir del reloj de temporización con el módulo *PulseGen*. Este pulso es recibido por el sincronizador y provoca el cálculo de los promedios de los valores recibidos (asociados a cada una de las cuatro variables) desde el último pulso y se envíe un paquete unificado con los resultados a los *SSDs*.
- Pulso de dato generado en los *Drivers*: cada vez que se produce un nuevo valor en los *Drivers* se genera un pulso que informa al sincronizador.
- Pulsos del sincronizador: cada paquete de valores generado va acompañado de un pulso de *Ready* para que los componentes de los *SSDs* realicen las operaciones oportunas. Posteriormente se genera un segundo pulso para que el *Predictor* produzca las predicciones oportunas con los datos resultantes. Además, este segundo pulso avisará también al *BuffersHandler* de la presencia de nuevos datos procesados (añadiéndolos por tanto a los correspondientes *buffers*).

3.5. Integración final

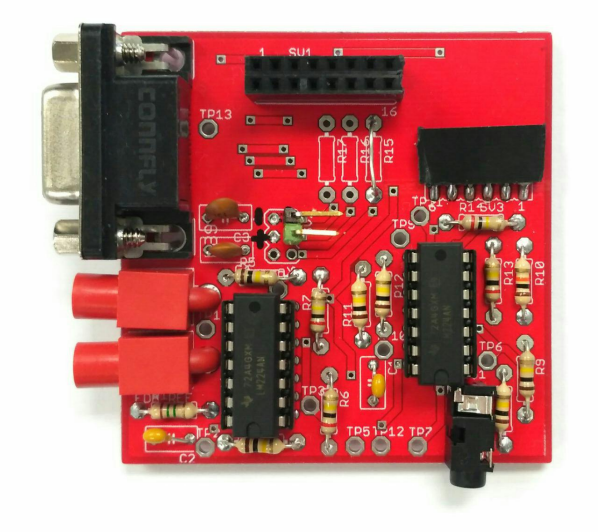


Figura 3.20: Placa para la gestión de sensores

Para la integración de los sensores con la *FPGA* se usa una placa adicional que recoge

la gestión de los mismos y sirve como interfaz. Como se puede ver en la imagen 3.20 esta placa cuenta con un conector DB9 (para la entrada del pulsioxímetro), un conector Jack de 3.5mm (para el sensor de temperatura) y dos conectores Snap (para la entrada de los datos de sudoración).

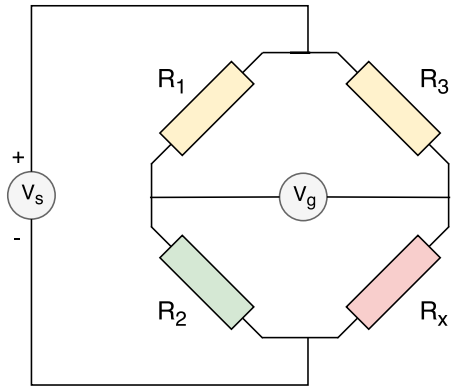


Figura 3.21: Puente de Wheatstone

$$V_g = (\frac{R_2}{R_1 + R_2} - \frac{R_x}{R_x + R_3}) * V_s \tag{3.1}$$

Para obtener los valores de sudoración se debe medir la conductancia entre los dos electrodos. Para ello se usará un puente de Wheatstone (figura 3.21), con el que se puede medir una diferencia de tensión dependiente de la variación de impedancia de un sensor en una de las ramas de un puente balanceado. Se parte de una situación de equilibrio, en la que hay una caída de tensión nula en V_g. Para ello se debe escoger una resistencia de referencia en R_x (resistencia a medir) y calcular las otras tres. Posteriormente, al variar R_x el voltaje leído en V_g seguirá el comportamiento especificado en la ecuación 3.1. Tras despejar R_x se podrá conocer la conductancia (correspondiente a la inversa de la resistencia).

En el apéndice C se pueden observar el esquemático del circuito.

3.6. Síntesis

El proyecto se ha configurado teniendo como dispositivo objetivo la familia Zynq XC7Z010 (paquete CLG400). Además, dado que la placa dispone de DSPs, se ha configurado la opción *use_dsp48* para que use el número máximo de estos módulos para reducir el área resultante y aumentar la eficiencia (estando la opción *dsp_utilization_ratio* a 100). Se pueden observar las opciones de HDL en la figura 3.22.

Por otra parte, de forma previa a la síntesis del proyecto, se pueden variar los parámetros de operación desde los paquetes de tipos y constantes definidos en el proyecto (figura). Algunos de los parámetros principales que se pueden encontrar en estos paquetes son los siguientes:

- Tamaño de los tipos de datos usados para coma fija, diferenciando el tipo de dato usado para la transmisión de datos (*FIXED_STD*) y el usado para la realización de operaciones (*FIXED_EXT*).
- Parámetros de operación de los módulos de detección de errores (saturación, caída y ruido) para cada una de las variables, así como el tamaño de los *buffers* con los que operan.
- Tiempo que se debe prolongar un error para descartar una variable.
- Intervalo de tiempo para la generación de pulsos al sincronizador (por defecto un minuto).
- Límite por encima del cual se considera que una predicción indica la presencia de un futuro episodio de dolor.

Los resultados de la síntesis para los parámetros elegidos se pueden observar en la tabla 3.2.

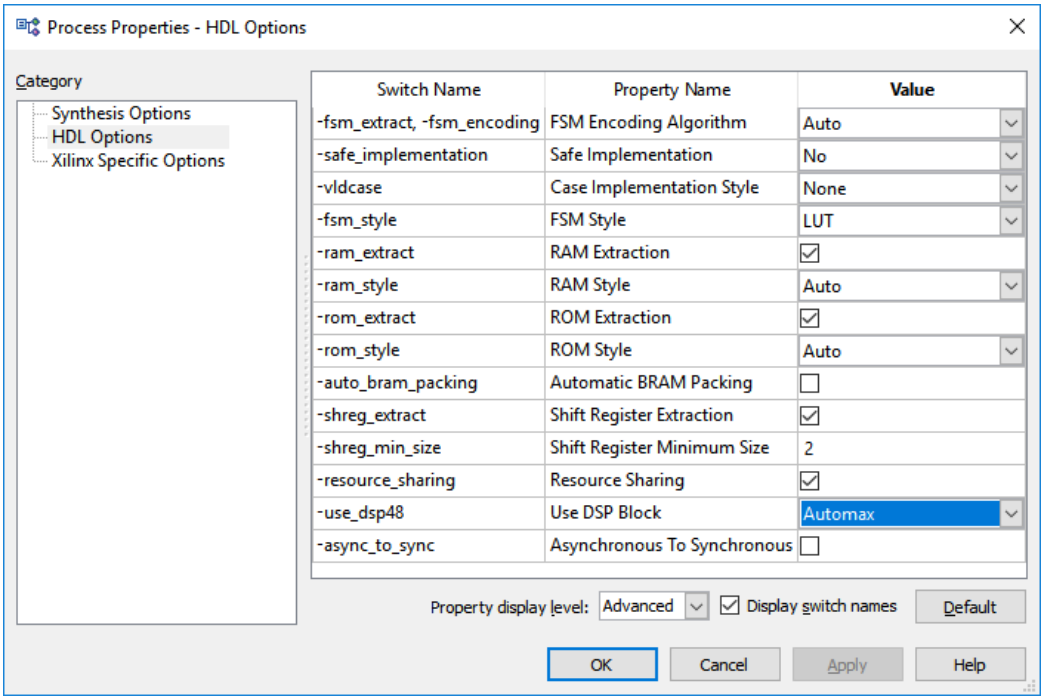


Figura 3.22: Opciones de síntesis (HDL) establecidas en Xilinx ISE

Cuadro 3.2: Resultados de la síntesis

	Consumidos	Disponibles	Porcentaje (%)
Slice Registers	9615	35200	27
Slice LUTs	13070	17600	74
IOBs	14	100	14
BUFG	8	32	25
DSP48E1s	80	80	100

Capítulo 4

Evaluación

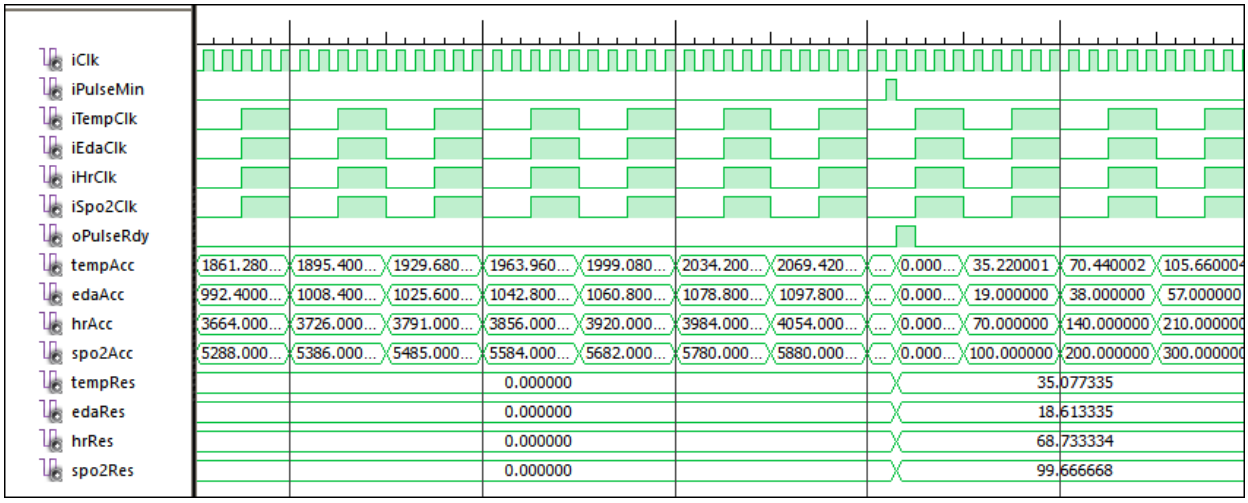
La evaluación de la implementación se ha realizado en dos fases. Primeramente, se efectúa una simulación software de cada uno de los módulos VHDL, comprobando con bancos de prueba su correcto funcionamiento (con el programa *ISim*). Tras ello, se comprueban las funcionalidades características del sistema directamente sobre la FPGA, de forma que quede validado.

4.1. Simulación de comportamiento

Como primera etapa en la evaluación del sistema de predicción se han creado una serie de bancos de prueba (testbenchs) en los que se simulará el comportamiento de cada módulo con distintas entradas que permitan comprobar la correcta operación de sus funcionalidades. Las salidas aportadas por cada uno de ellos se comparan con resultados correctos conocidos (usando aserciones, con la palabra clave *assert* de *VHDL*).

4.1.1. Sincronizador

Por otra parte, en la figura 4.1a se puede observar como, a medida que van llegando nuevos datos al sincronizador, se suman en los acumuladores (*tempAcc*, *edaAcc*, *hrAcc*, *spo2Acc*) y cuando llega el pulso que informa de la finalización de un minuto se realiza



(a) Acumulación de los datos entrantes

(b) Generación de pulsos de salida

Figura 4.1: Simulación del sincronizador

el promedio de cada uno de los datos y se genera un paquete de salida que agrupa estos cuatro datos.

En la figura 4.1b se muestra en detalle el comportamiento del sincronizador tras la llegada del pulso de entrada. Para gestionar este proceso este componente se apoya en una máquina de estados (representada en la figura 4.2), que consta de los siguientes estados:

- *IDLE*: estado en el que se encuentra el sincronizador no está generando pulsos de salida. Cuando llega el pulso que marca el fin del minuto actual, además de generar el paquete con los resultados, se activa la señal *oPulseRdy* (para informar a los *SSD* de la disponibilidad de este nuevo valor) y se pasa al siguiente estado.
- *SRDY_LOW*: se pone a nivel bajo la señal *oPulseRdy* y se pasa al siguiente estado.
- *WAITING*: estado que tiene como función esperar a que se regeneren las señales en los *SSD* (en caso de ser necesario). Por tanto, los ciclos de espera coinciden con los ciclos de operación de la generación de resultados en el *ARX*. Para llevar cuenta de los ciclos esperados se apoya en la variable *cont*. Cuando este contador llega al número de ciclo deseado (*NC*) se pasa al siguiente estado.
- *PRDY_HIGH*: se activa la señal *oPulsePred*, que activa la operación del predictor. Además, también despertará al *BuffersHandler*. Tras esto pasa inmediatamente al siguiente estado.
- *PRDY_LOW*: pone a nivel bajo la señal *oPulsePred* y vuelve al estado de reposo (*IDLE*).

4.1.2. Detectores de errores

En la figura 4.3 se muestra el comportamiento de los tres detectores de errores. Todos ellos mantienen un conjunto de los últimos valores pasados de la variable que están

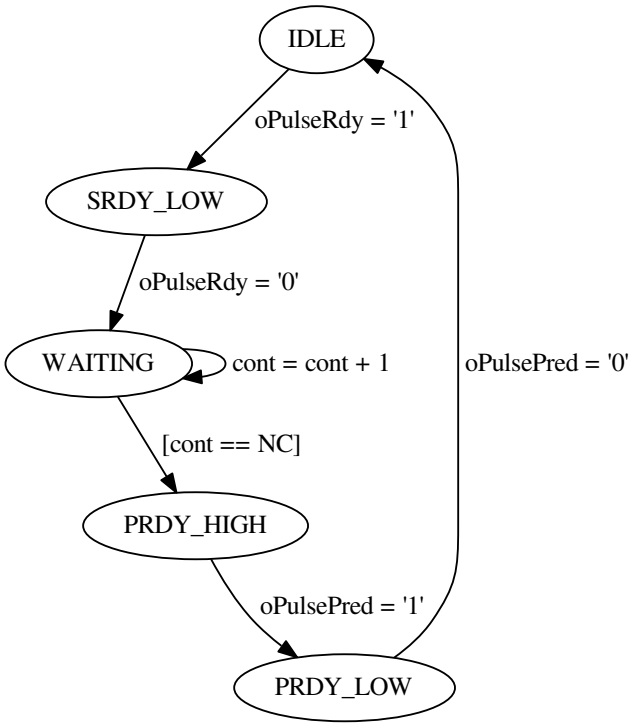
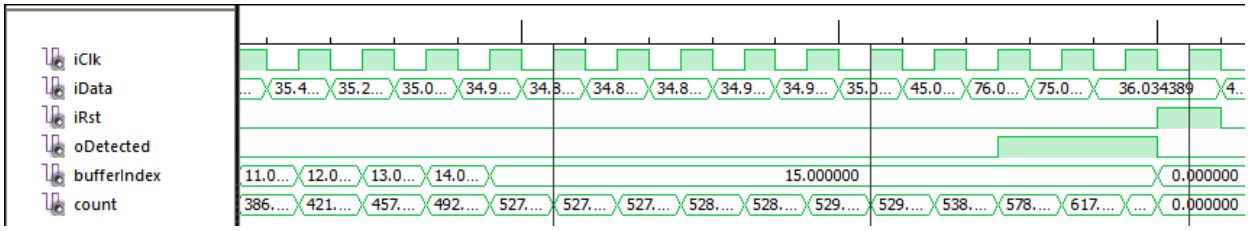


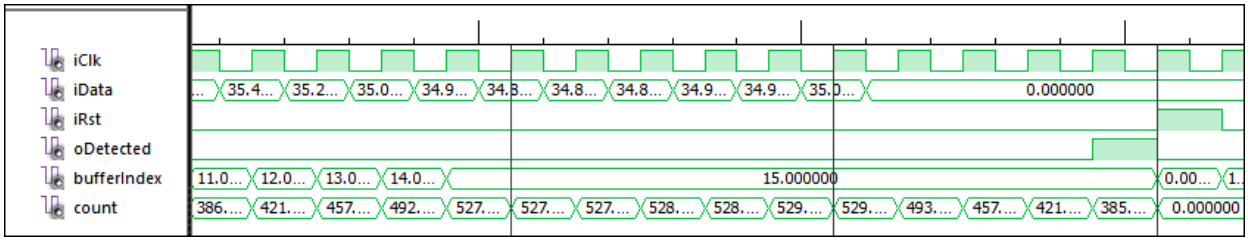
Figura 4.2: Grafo de estados del sincronizador

monitorizando (15 en el ejemplo). La variable *bufferIndex* identifica el índice de la siguiente posición del *buffer* a rellenar durante el proceso de inicialización. Una vez está lleno (cuando llega a 15) se mantiene constante este valor hasta que se reinicie el componente (momento en el cual se pone a 0 y se vuelve a realizar el procedimiento de relleno de datos). Además, se mantiene un contador con la suma de los valores presentes en dicho *buffer* (*count*), en el que se mantendrá una suma de los últimos valores que se hayan leído. Durante el proceso de inicialización, la salida se mantiene a nivel bajo (no se informa de ningún error). Cuando el *buffer* está lleno empiezan las comprobaciones, que varían en función del módulo que se observe:

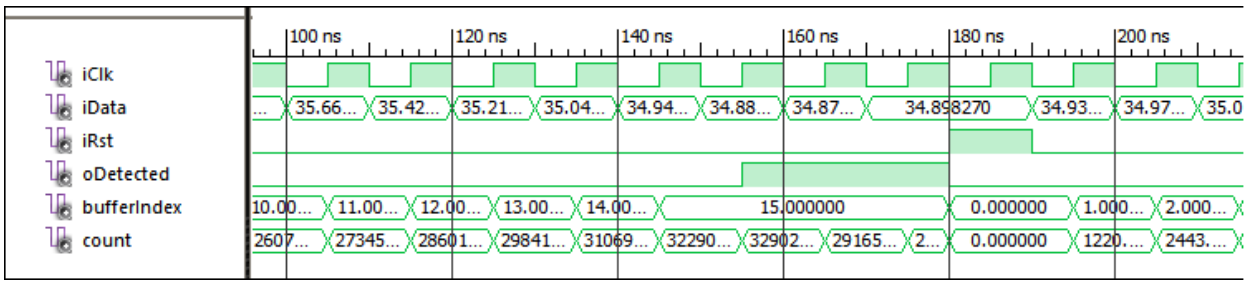
- El detector de saturaciones (figura 4.3a) informará de un error cuando el promedio de los valores registrados supere un determinado umbral (38 en el ejemplo).



(a) Detector de saturaciones



(b) Detector de caídas



(c) Detector de ruido

Figura 4.3: Simulación de los componentes de detección de errores

- El detector de caídas (figura 4.3b) informará de un error cuando el promedio de los valores registrados sea inferior a un determinado umbral (5 en el ejemplo).
- El detector de ruido (figura 4.3c) informará de un error cuando el promedio de los valores registrados al cuadrado sea superior a un determinado umbral (1600 en el ejemplo).

Cada uno de ellos cuenta además con una señal de Reset (*iRst*), que reinicia la operación de los detectores de errores. Esta situación se puede observar en las simulaciones asociadas, en las que al activarse esta señal cuando la salida está activa se anula la misma y se reinician las señales *bufferIndex* y *count*. Esta señal será dependiente de un botón físico que el usuario deberá pulsar cuando se haya informado de un error y ya se haya corregido (para agilizar la restauración de los componentes del *SSD*.

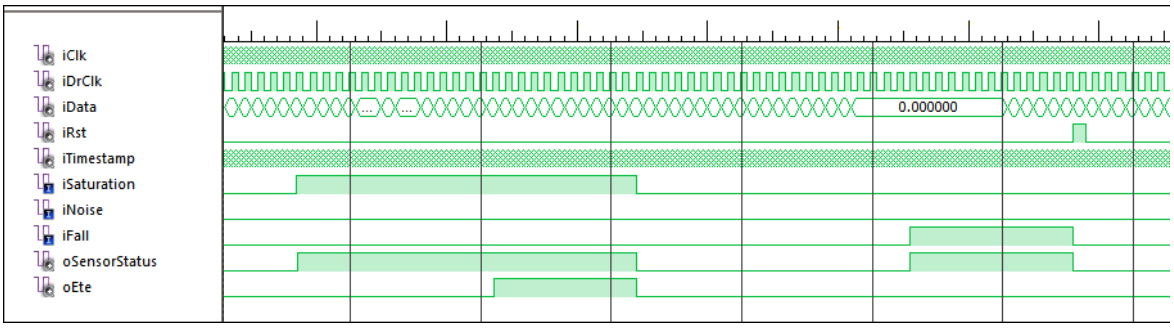


Figura 4.4: Simulación del componente de detección de anomalías (*AnomalyDetector*)

El módulo de detección de anomalías, que tiene como entradas las salidas generadas por estos tres componentes, debe activar la señal *oSensorStatus* cuando al menos uno de los tres haya informado de un error y activar la señal *oETE* cuando la presencia de errores se prolongue en el tiempo un intervalo previamente definido. En la figura 4.4 se puede observar como al activarse el error de saturación se levanta la señal *oSensorStatus*. Como el intervalo se mantiene durante el tiempo preestablecido también se activa la señal *oETE*. En este caso esta situación se mantiene hasta que el detector de saturaciones deja de percibir errores en los datos entrantes, momento en el cual se reinicia el estado del detector de anomalías (poniendo a nivel bajo sus dos salidas).

Posteriormente, se detecta una caída en la señal. Sin embargo, en este caso no se llega a poner a nivel alto la señal *oETE*, ya que antes de que transcurra el tiempo necesario para ello se levanta la señal de reset, que reinicia el estado de los componentes (teniéndose que volver a llenar los *buffers* de los detectores de errores del *SSD* con el que estén relacionados antes de detectar nuevas anomalías).

4.1.3. ARX (regeneración de señales)

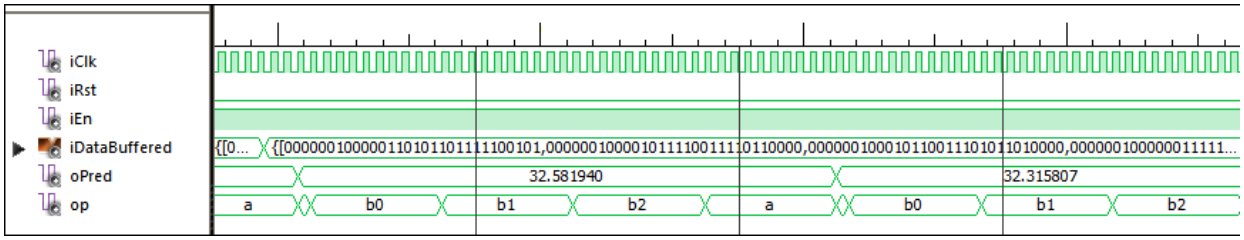


Figura 4.5: Simulación del componente *ARX*, que recupera de señales en caso de caídas

En el caso de que se detecte alguna anomalía entra en funcionamiento el *ARX* (figura 4.5), que genera nuevos datos de la variable que ha sufrido el fallo en base a valores pasados (tanto de esa variable como de las exógenas). Estos datos son suministrados por el *BuffersHandler*, componente que recoge los datos de la salida del sincronizador y que centraliza la recogida de datos pasados. En la figura 4.7 se pude observar como en cada pulso entrante va actualizando los *buffers* de cada una de las variables. De esta forma, al rellenarlos por completo (llegando a los 10 datos en el ejemplo) se activa la señal *oFullBuffer*, informando sobre la posibilidad de utilizar esta información a los *SSDs*.

El proceso de regeneración de datos se apoya en la máquina de estados representada en la figura 4.6. En ella se ve como al activarse el proceso se pasa por tres estados BX (*B0*, *B1*, *B2*), correspondientes al cálculo asociado a las tres variables exógenas. Posteriormente se pasa por el estado *A*, correspondiente al cálculo asociado a las entradas pasadas de la variable a calcular. Esta última tendrá un impacto mayor que

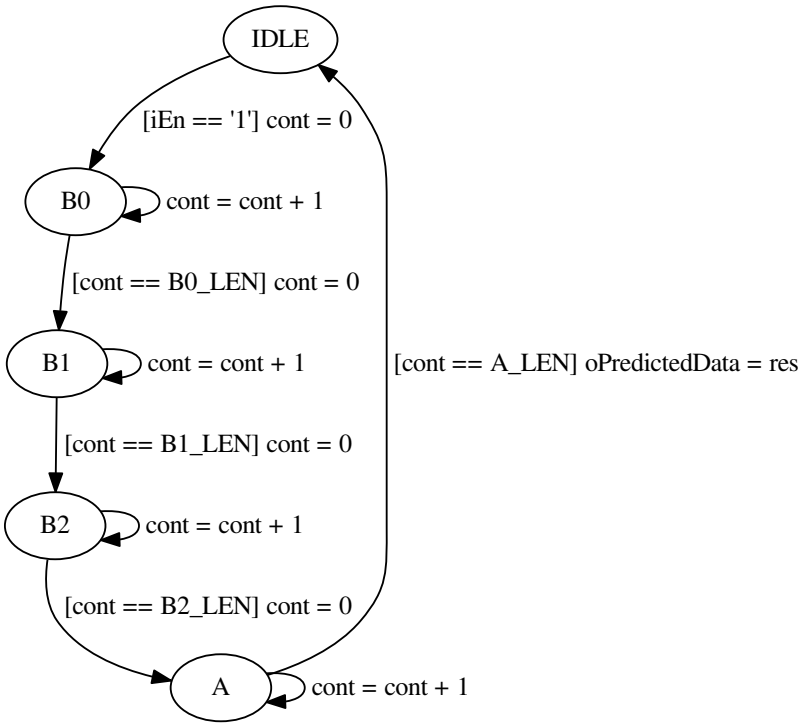


Figura 4.6: Grafo de estados del *ARX*

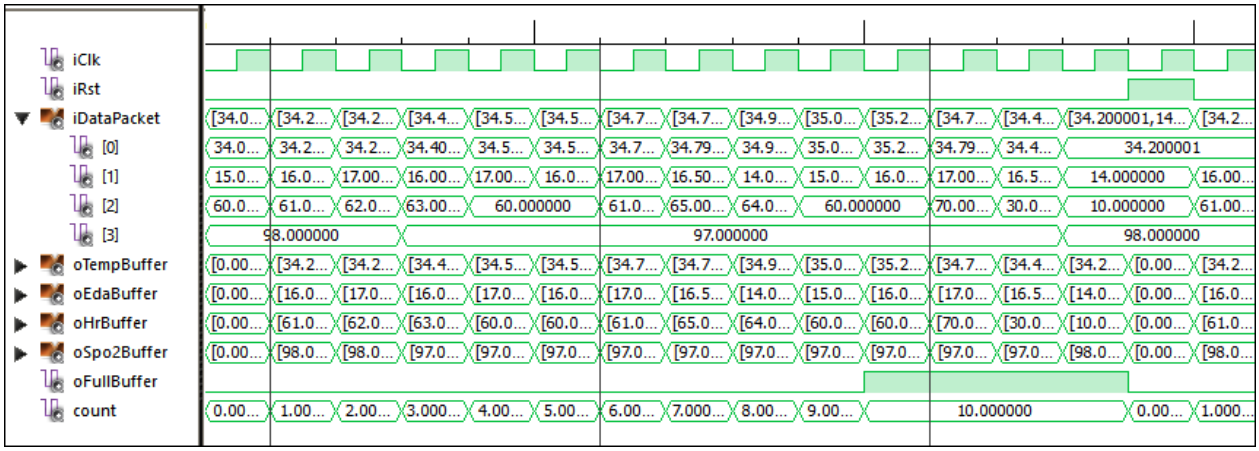


Figura 4.7: Simulación del componente *BuffersHandler*, que agrupa valores pasados de las cuatro variables

las otras tres, ya que tiene correspondencia directa con la variable a predecir. El cálculo

asociado a cada una de estas etapas corresponde a la multiplicación de los vectores de datos pasados por otros precalculados (generados al crear el modelo).

4.1.4. Predictor

Como se ha comentado anteriormente, el *Predictor* está formado por tres componentes: el *SDMS2*, que gestiona el proceso de generación de predicciones, el *SSME*, que calcula las predicciones en base al estado actual y los datos de entrada, y el *LinearCombiner*, que agrupa las predicciones obtenidas para generar un único valor.

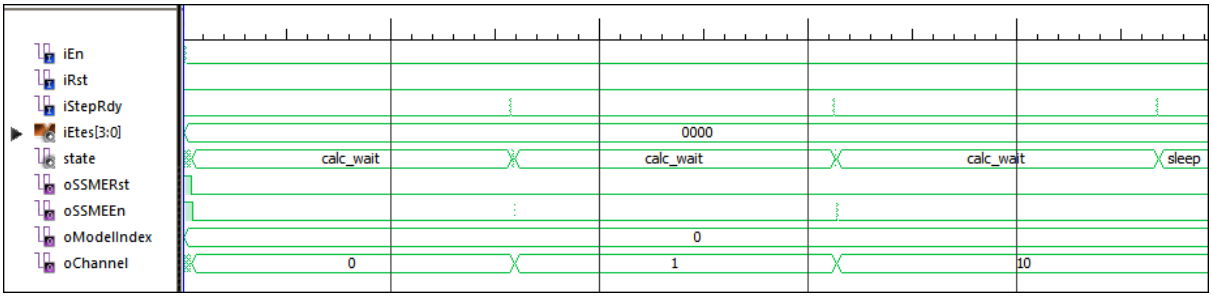


Figura 4.8: Simulación del *SDSM2*, encargado de controlar la operación del *SDSM2*

En la figura 4.8 puede observarse como el *SDSM2* manda peticiones al *SSME* para generar predicciones. Para ello, establece los modelos y el canal adecuados en función de la situación (tal como se explicó anteriormente). Cada una de estas peticiones se inicia con un pulso en la señal de Enable (*iSSMEEn*). Tras este pulso, el *SDSM2* manda una señal para iniciar el cálculo de una predicción en el *SSME* (estado $CALC\_REQ$) y queda a la espera (con el estado $CALC\_WAIT$) de que el *SSME* finalice sus cálculos, hecho que se percibe cuando se genera un pulso en la señal de ready (*iStepRdy*). En el momento en el que no deba hacer más peticiones (cuando se hayan generado las tres del conjunto actual) el *SDSM2* entrará en modo de reposo (estado *SLEEP*), en el que permanecerá hasta que sea despertado por un nuevo pulso del sincronizador. Este comportamiento se ve ilustrado en forma de grafo en la figura 4.9.

En la imagen relativa a la simulación del *SSME* (4.10) se puede observar su operación

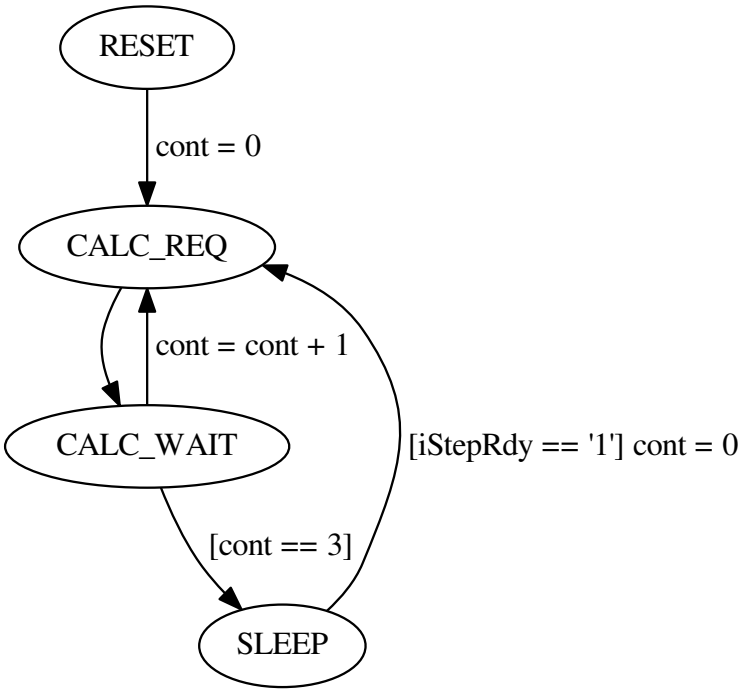


Figura 4.9: Grafo de estados del *SDSM2*

Figura 4.10: Simulación del *SSME*, encargado de la generación de predicciones basándose en espacio de estados

para cada una de las peticiones de cálculo entrantes. En el estado *pred* este componente generará la predicción basándose en el estado actual. En los estados NS_F y NS_S se realizan las operaciones (multiplicaciones de matrices) necesarias para la generación del siguiente estado. Después de cada predicción se pasa por un estado para dar salida

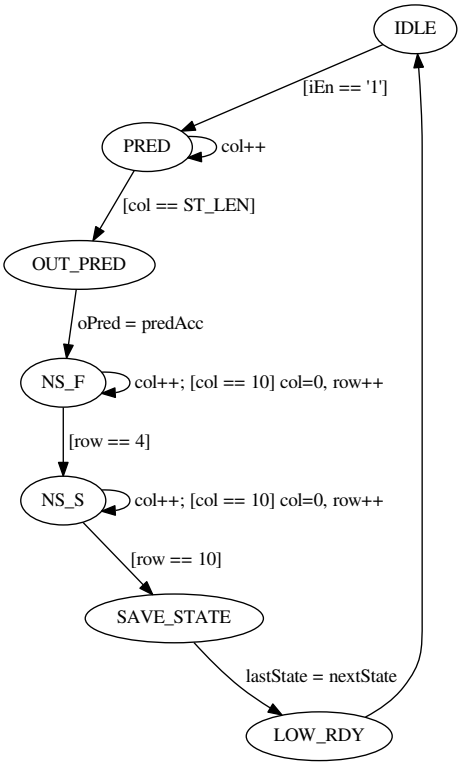


Figura 4.11: Grafo de estados del *SSME*

al valor generado. Tras cada generación de estado se almacena el resultado para la siguiente iteración y se genera una señal de ready (*oRdy*) para informar al *SDSM2* del fin de la operación. Este comportamiento se ve reflejado en el grafo de la figura 4.11. Por otra parte, cuando se recibe una señal de reset *iRst* se itera entre cada uno de los tres canales disponibles, reiniciando el estado en cada uno de ellos para futuros cálculos.

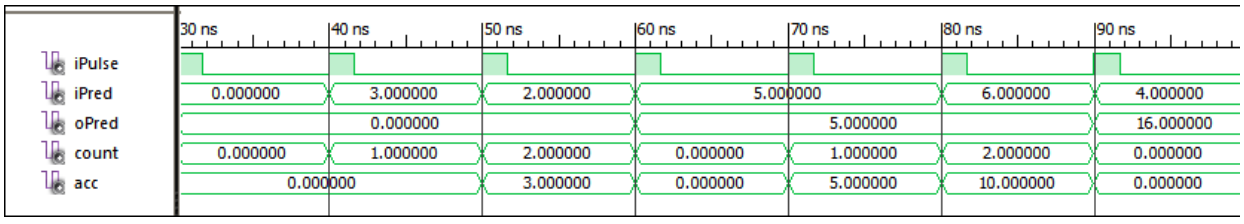


Figura 4.12: Simulación del *Linear Combiner*, que agrupa las predicciones del *SSME*

A medida que se van generando predicciones en el *SSME*, se van acumulando en el componente *LinearCombiner* para realizar posteriormente el promedio. Esto se ve representado en la figura 4.12, donde se ve como se suma cada una de las tres predicciones de cada grupo en un acumulador y se proporciona como salida el resultado. La aceptación de nuevas predicciones es controlada mediante la recepción de pulsos desde el *SSME* (señal *oRdy*). Como en ejemplos anteriores, para reducir el número de divisiones (y dado que el número de elementos es fijo) el promedio se interpretará en base a la suma de los valores (comparando el valor de la suma con el valor deseado multiplicado por el número de elementos).

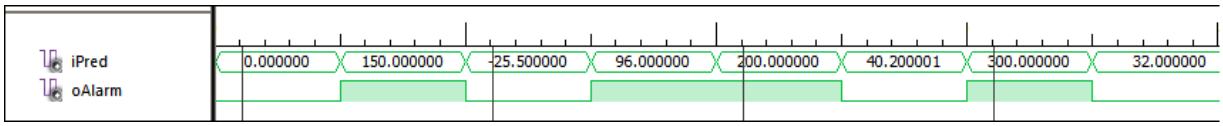


Figura 4.13: Simulación del componente decisor (*Decider*)

Posteriormente, la salida del *LinearCombiner* se dirige al *Decider* (figura 4.13), componente combinacional encargado de comprobar si la predicción combinada que se le proporciona supera un límite preestablecido. Por tanto, dado que se quiere saber si el promedio es superior a 32 y el valor de entrada representa la suma de tres predicciones, se generará un valor alto en la señal de alarma cuando el valor entrante sea igual o superior a 96 (32*3).

4.1.5. Simulación completa

Por último, en la figura 4.14 se puede observar una simulación del componente raíz del sistema de predicción. En ella, se pasan distintos valores de las cuatro variables biométricas de las que depende el sistema (correspondientes a un paciente real) y se observa como, cuando el valor supera el límite (32 en el ejemplo, considerando la media de las tres predicciones) se genera la correspondiente alarma. De la misma forma, cuando la predicción vuelve a ser inferior a ese valor la señal de alarma retorna

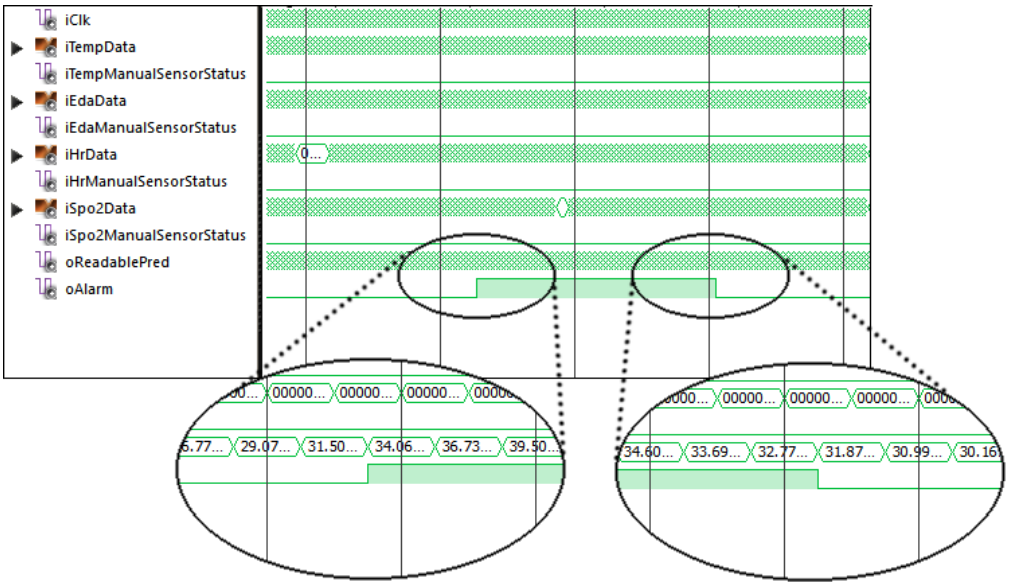


Figura 4.14: Simulación del componente raíz del predictor de migrañas

a nivel bajo.

4.2. Entorno de evaluación

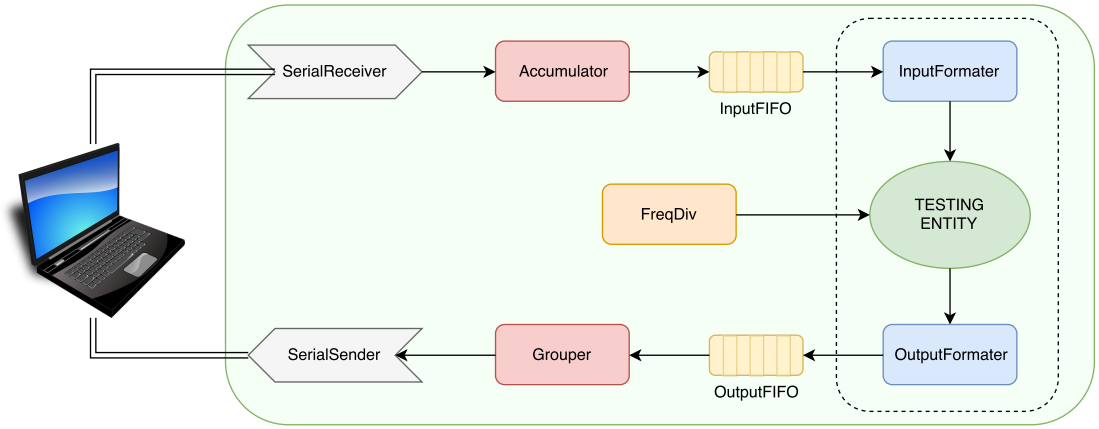


Figura 4.15: Estructura del entorno de evaluación

Para la comprobación del sistema se ha creado un entorno de evaluación que se aloja en la FPGA y permite alimentar al sistema con los datos recibidos mediante protocolo serie[11] (usando como interfaz un adaptador serie USB como el que se puede ver en la

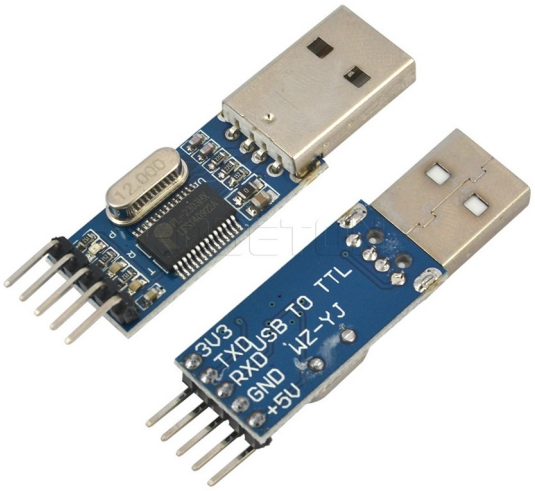


Figura 4.16: Adaptador serie a USB

figura 4.16). De esta forma, un controlador se encargará de codificar los datos para enviarlos al entorno. Una vez recibidos, el entorno les aplicará el formato adecuado, los cargará en el sistema y enviará de vuelta los resultados. Para controlar las diferencias entre los tiempos de operación y la llegada de datos se añaden unas colas[1] en las que se depositen los datos entrantes pendientes y los resultados generados (a la espera de ser procesados en el momento oportuno).

La estructura del sistema de evaluación puede verse en la figura 4.15 y consta de los siguientes componentes:

- SerialReceiver: Se encarga de recibir los datos enviados desde el PC por protocolo serie, proporcionando como salida el byte actual y una señal que indica cuando se ha recibido un nuevo byte.
- Accumulator: se encarga de agrupar un determinado número de bytes, que representen un conjunto de valores para las entradas de la entidad a evaluar. Por tanto, el tamaño dependerá de los tipos de datos de la entrada y de la codificación empleada para transmitir los datos a la FPGA por el puerto serie (pudiendo ser

necesario añadir unos bits de relleno).

- InputFIFO: cola en la que se almacenan los conjuntos de datos proporcionados por el acumulador. De esta forma se aisla la comunicación de datos con el sistema a evaluar, funcionando como entidades independientes.
- InputFormater: se encarga de formatear un conjunto de bytes para adecuarlos a la entrada se espera recibir en el sistema a evaluar.
- Testing Entity: componente raíz del sistema a evaluar.
- OutputFormater: se encarga de formatear la salida del sistema, codificándola y agrupándola en un conjunto de bytes.
- OutputFIFO: cola en la que se almacenan los resultados del sistema evaluado.
- Grouper: recoge los resultados de la cola de salida y los va enviando byte a byte al controlador serie.
- SerialSender: envía los datos producidos al ordenador usando un protocolo serie.
- FreqDiv: genera una frecuencia de reloj adecuada para el correcto funcionamiento de la entidad a evaluar (a partir de la señal de reloj principal de la FPGA).

4.3. Validación en placa de desarrollo

Una vez simulados los distintos componentes del sistema y desarrollado el entorno de evaluación se procede a validar los módulos más significativos del sistema, de forma que el sistema quede validado y pueda ser usado para estudios futuros.

En las distintas pruebas que se presentan a continuación se usan scripts de *Python* para comunicarse con el puerto serie e inyectar datos controlados en los módulos implicados, de forma que puedan comprobarse los resultados. En los casos en los que sea necesario se desarrollará una implementación software paralela de forma que se comparen sus resultados con los devueltos por el sistema.

4.3.1. Entrada de datos

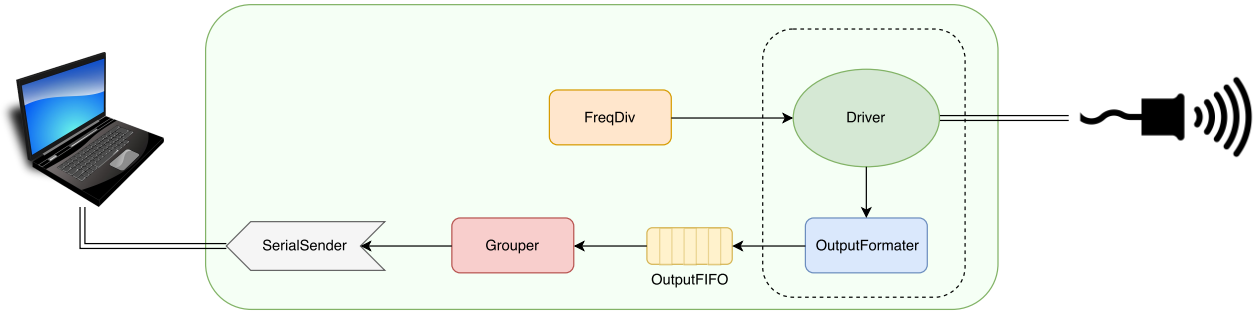


Figura 4.17: Estructura del entorno de evaluación (para *Drivers*)

Para comprobar el correcto funcionamiento de los sensores y los *Drivers* se han realizado validaciones separadas con cada uno de ellos, de forma que se envíen lecturas de los sensores a intervalos regulares usando el entorno previamente creado. Sin embargo, en este caso (dado que no se necesita enviar datos a la FPGA desde el ordenador), únicamente se usan los componentes de formateo, codificación y envío de datos (además de los componentes a validar y los componentes de generación de señales de reloj). Por tanto, la estructura del entorno usado corresponde al de la figura 4.17.

Para comprobar el correcto funcionamiento del *Driver* encargado de recoger los datos de saturación de oxígeno y ritmo cardíaco se ha introducido en el entorno de evaluación y se ha configurado para generar datos cada segundo. Estos datos se envían por conexión serie y son formateados e interpretados por un script Python. Para la recogida de datos se ha usado el sensor con adaptación para dedo mostrado en la imagen 3.9a. En la figura 4.18 se puede visualizar la recepción de esos datos y como, tras colocar el sensor correctamente, pasa de modo desconectado a enviar lecturas válidas.

Por otra parte, y de forma similar al anterior, se comprueba el *Driver* encargado de recoger y formatear los datos de temperatura y sudoración. Este componente interpreta los datos que le llegan por *I2C* desde el conversor analógico/digital *PModAD2*. En la captura 4.19 se puede observar como llegan estos datos a través de la conexión serie del entorno de evaluación, leyendo este par de datos cada 3 segundos.

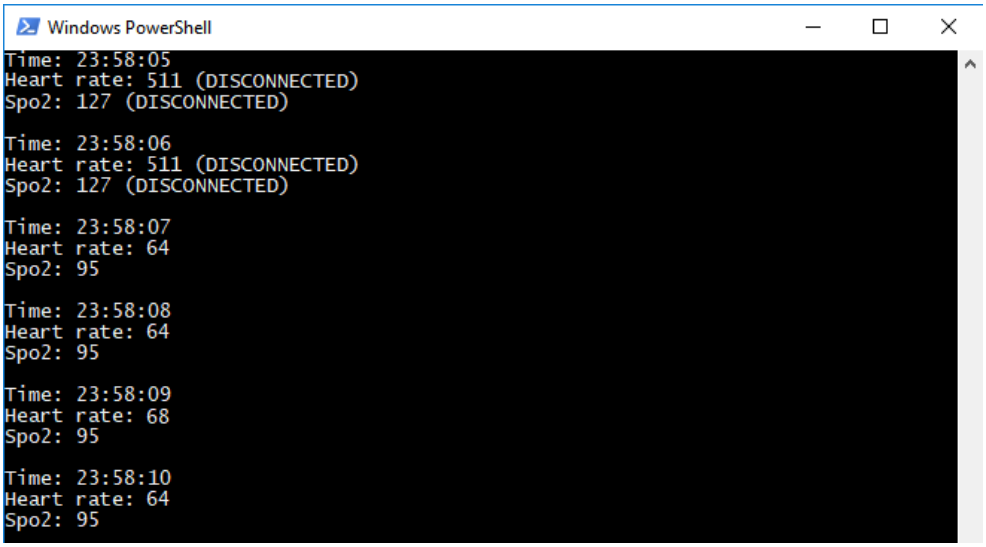


Figura 4.18: Ejecución del script de validación del *HRSpo2Driver*

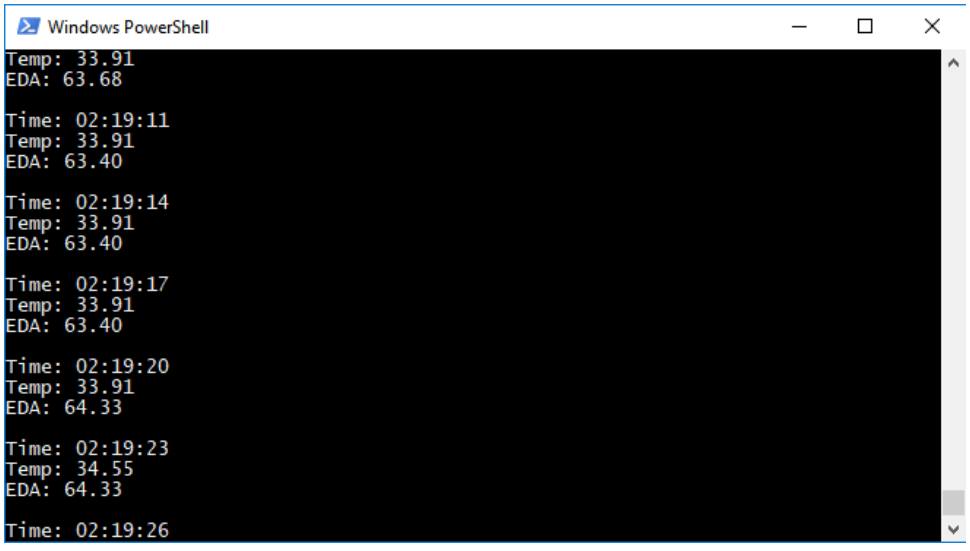


Figura 4.19: Ejecución del script de validación del *TempEdaDriver*

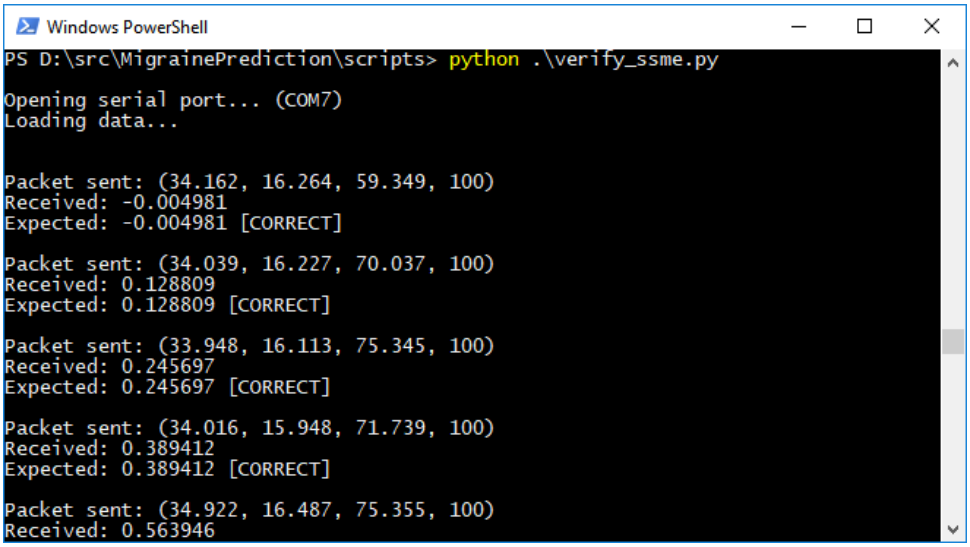


Figura 4.20: Ejecución del script de comprobación del *SSME*

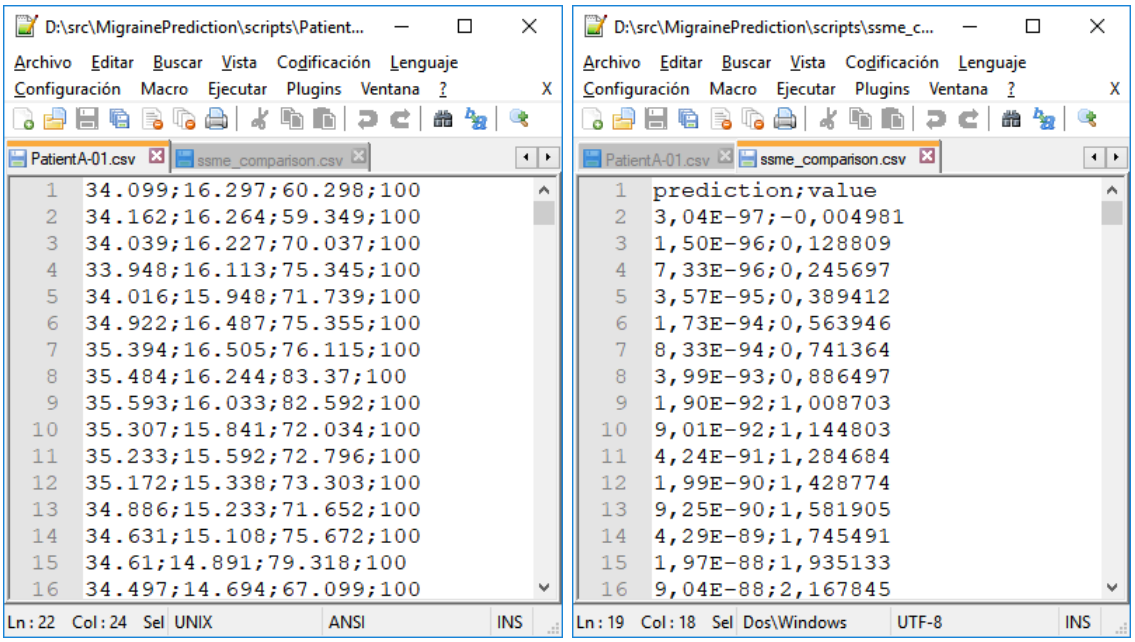
4.3.2. Generación de predicciones (SSME)

Para comprobar la generación de predicciones válidas se introducen los datos de un paciente real (figura 4.21a) en el evaluador de modelos, de forma que se obtenga una curva de dolor en base a los modelos previamente entrenados. Para ello, mediante un script desarrollado en Python se van introduciendo valores y comprobando con los valores esperados (precalculados). La ejecución de este script se puede observar en la figura 4.20, donde se comprueban los correctos resultados del sistema.

Por otra parte, se cuenta también con los niveles de dolor reales reportados por el usuario mediante la monitorización. La comparación de los niveles reales y predichos puede observarse en la figura 4.22.

4.3.3. Detección de errores (SSD)

Para esta prueba, se introducirá el componente *SSD* en el entorno de validación, de forma que se extraigan por separado las detecciones de errores de saturación, caídas y ruido, así como la señal que indica la continuación en el tiempo de errores en la variable a monitorizada (*ETE*). Por otro lado, se calculará esta generación mediante



(a) Datos de variables biométricas (b) Datos de dolor (real y predicho)

Figura 4.21: Fuentes de datos para la comprobación del *SSME*

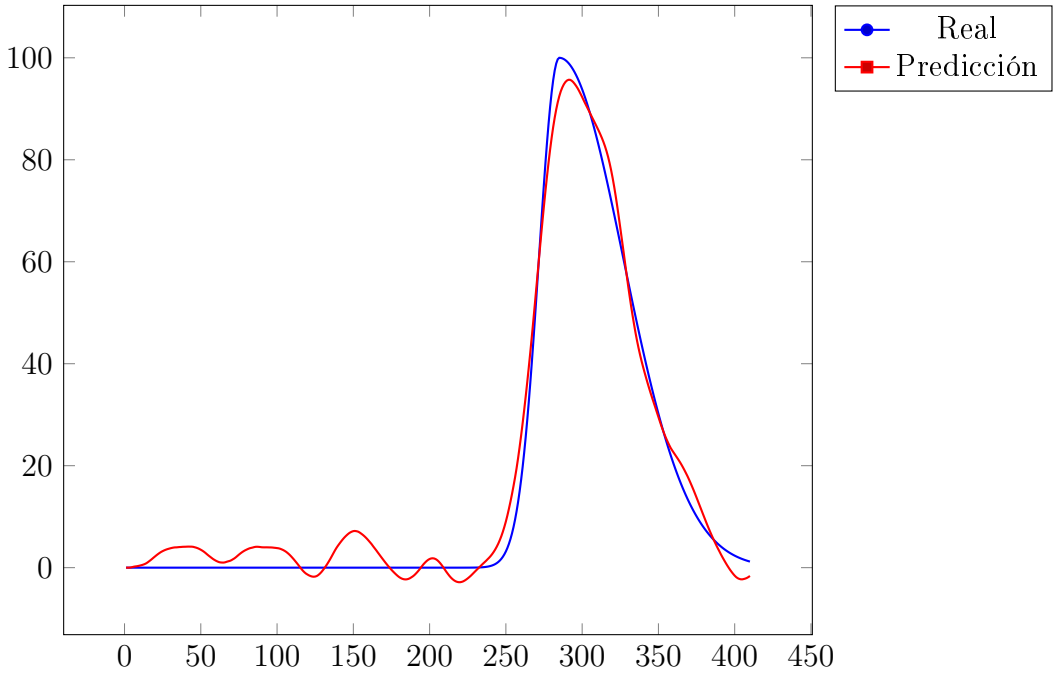


Figura 4.22: Comparación de los niveles de dolor (real y predicho)

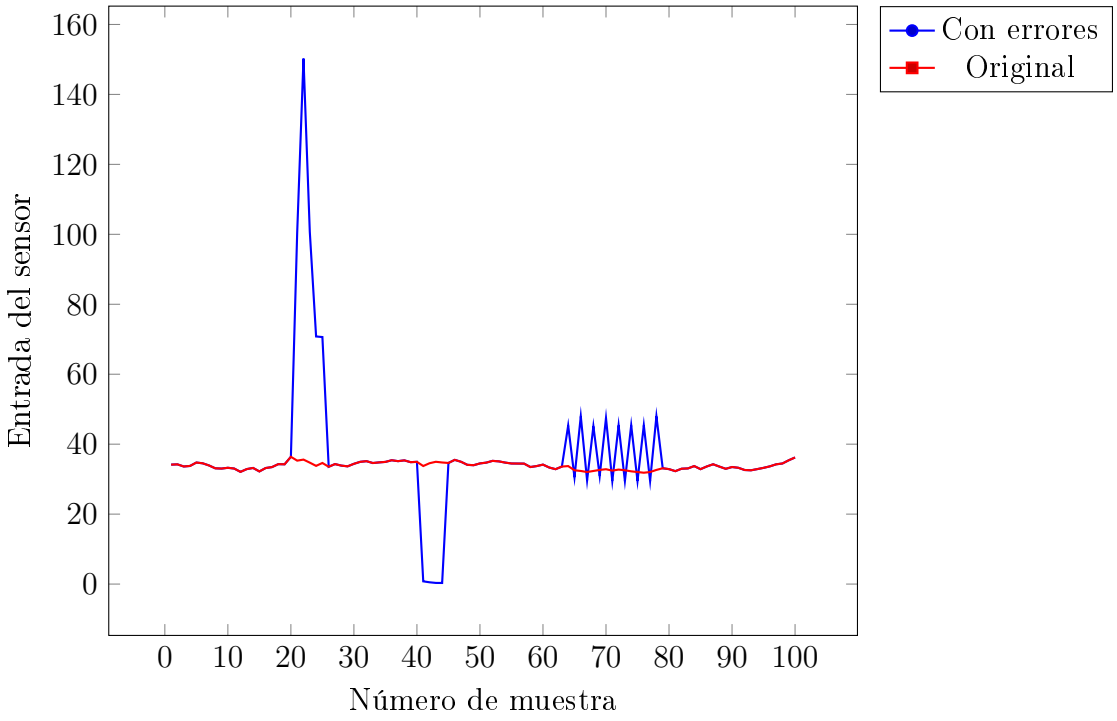


Figura 4.23: Valores de temperatura para la comprobación de errores

un script Python y se comparan los resultados a medida que se introduce información en la plataforma.

En concreto, se introducen los datos de temperatura visibles en la figura 4.23. En ellos se han introducido errores de todos los tipos detectados, de forma que se pruebe al completo esta funcionalidad:

- Hay una primera zona donde se deberá detectar una saturación, así como una alerta posterior de ruido.
- Una segunda zona representa una caída, manteniendo en valor 0 los valores producidos durante 4 muestras.
- Por último, se produce un ruido oscilante entre valores entre el intervalo $(29, 46)$, debiendo producir una alerta por detección de ruido (sin producirse saturación).

Para la realización de estas pruebas se han establecido las siguientes configuraciones:

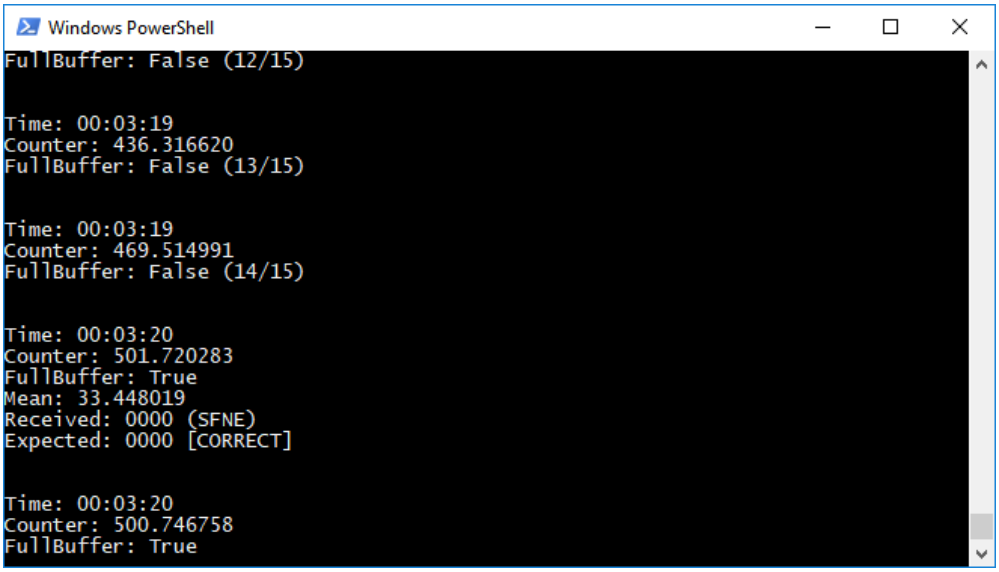
- Temperatura máxima: 38ºC.
- Temperatura de caída: 27ºC.
- Índice de ruido acumulado máximo: 20500.
- Tamaño de los *buffer* de detección: 15.
- Intervalo de tiempo para activar la señal *ETE*: 30 segundos.

La ejecución de esta prueba puede verse en la figura 4.24. En la captura 4.24a se ve como se inicializa el *buffer*, tras la carga de la plataforma en la FPGA, con los primeros datos de la lista de valores. Una vez inicializado, como en el sistema real, se comienza a buscar la presencia de fallas en la señal (manteniendo un *buffer* con los últimos valores y un contador con la suma de los mismos). En la captura 4.24a se muestra como se van comparando los datos generados por el sistema con los producidos por el script. Además, para probar la generación de la señal de *ETE* se han introducido retardos en el envío de datos de forma que se produzcan activaciones de la misma (en el ejemplo se activa al pasar 30 segundos, como se observa en las zonas remarcadas en rojo).

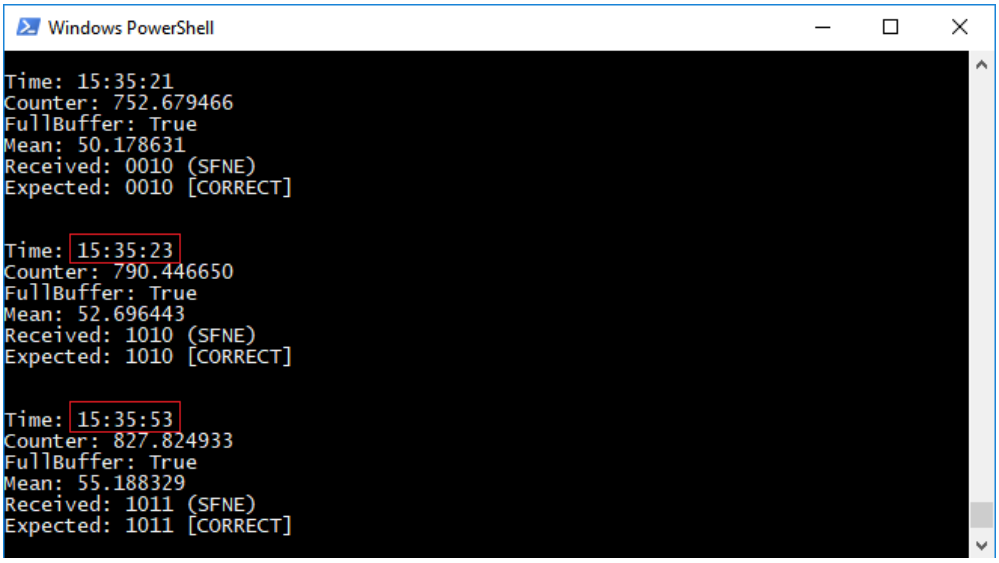
4.3.4. Regeneración de señales

Cuando un *SSD* comienza a detectar un fallo en la señal monitorizada el componente *ARX* debe generar estimaciones estadísticas a la espera de que se recupere la calidad de la misma o se anule debido a la prolongación del error. Para comprobar este comportamiento, se acoplará el *SSD* al entorno de evaluación y se provocan fallos en la señal. A modo de comprobación se comprobará la salida del SSD (mandando los datos por la salida serie del entorno), de forma que se observen los valores producidos (antes y después de producirse cada error).

En la figura 4.25 se observa como, en el punto en el que se informa internamente de una falla en la señal (error de saturación, en ese caso) se empiezan a producir valores en base a las entradas pasadas.



(a) Inicialización del script para la detección de errores



(b) Validación de los componentes de comprobación de errores (con activación de la señal ETE)

Figura 4.24: Ejecución del script para la comprobación de los componentes de detección de errores

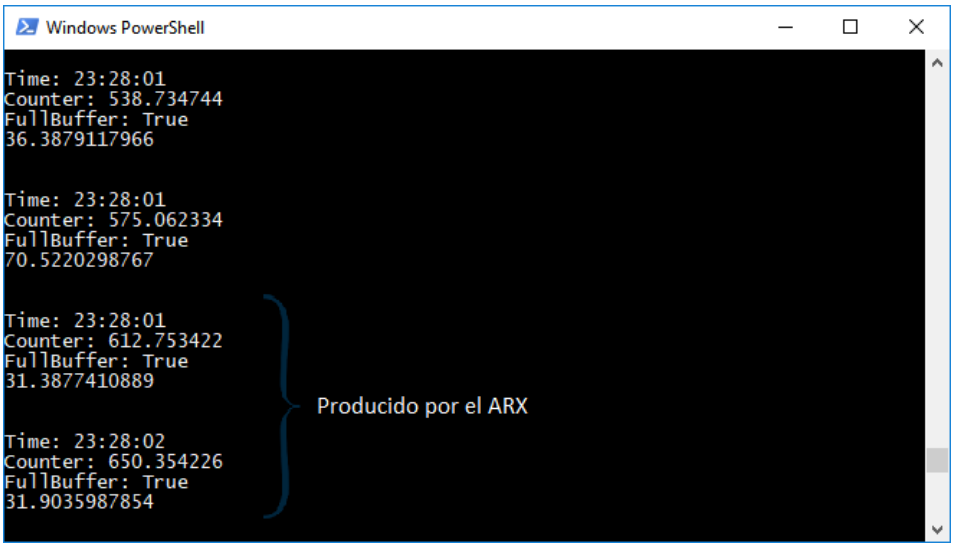


Figura 4.25: Valor producido por el *ARX* tras una falla en la señal

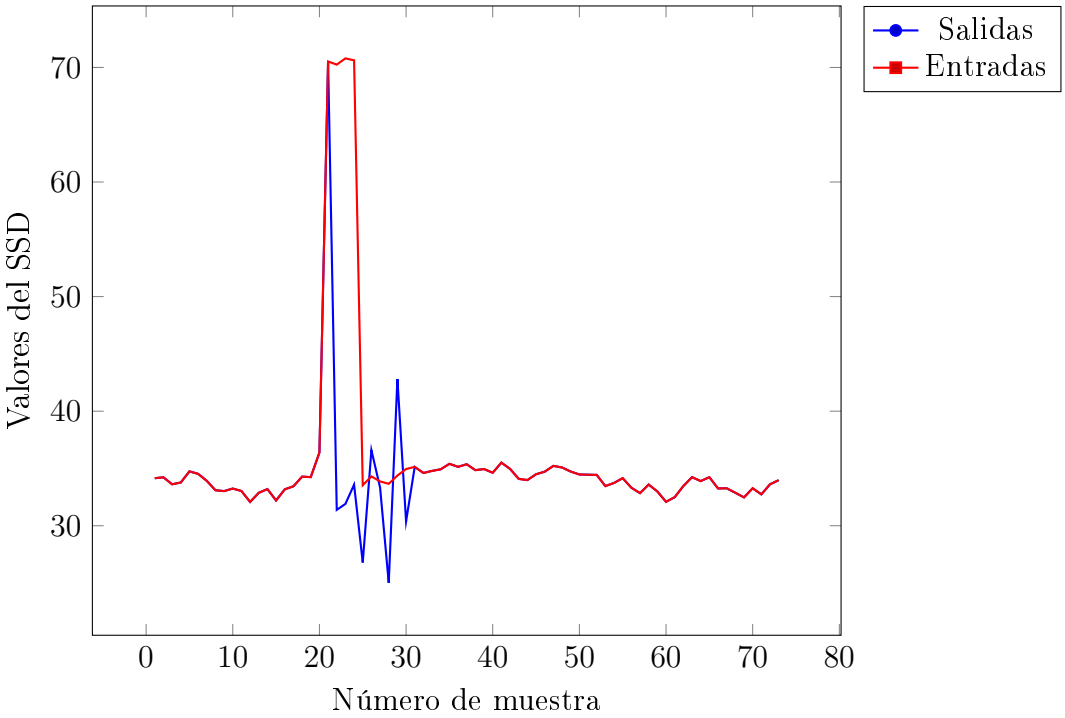


Figura 4.26: Salidas producidas en el *SSD* tras la generación de un error de saturación

En la figura 4.26 se muestran estos valores de forma gráfica. En color naranja se muestra las entradas reales registradas por el *SSD* en su entrada. En color azul, las salidas producidas por el mismo (teniendo en cuenta la regeneración). Como se puede observar, tras haberse producido la falla de saturación en la señal, el componente *ARX* se activa y regula la señal para que se aproxime a la tendencia anterior de la señal.

Capítulo 5

Conclusiones

En este proyecto se ha presentado un sistema de predicción de migrañas implementado en una *FPGA* mediante *VHDL*. Además, es importante recalcar que esta implementación parte de un entorno de simulación en un lenguaje formal (*DEVS*), lo cual permite validar de forma previa la plataforma y favorece el desarrollo.

El sistema maneja cuatro variables biomédicas que obtiene a través de diversos sensores (usando en dos de los casos un conversor A/D para adaptar la señal). Para la comunicación entre estos elementos se usa comunicaciones *I2C* y serie. Por otra parte, se monitoriza su calidad, de forma que se detecte cuándo se produce un fallo en alguna de ellas y se regeneren mediante técnicas de recuperación en caso de ser necesario. Posteriormente, con esta información, se generan predicciones de migraña siguiendo un modelo en el espacio de estados en función de las variables biométricas (cambiando los modelos de predicción en caso de que alguna de las señales falle). Además, el sistema está diseñado para poder añadir distintos mecanismos de predicción, de forma que se puedan generar estimaciones con distintos métodos y agrupar los resultados.

Tras el desarrollo del sistema, se ha evaluado su funcionamiento de dos maneras distintas. La simulación software de los distintos módulos del sistema de forma independiente ofrece una vista confiable de su funcionamiento. Posteriormente, mediante el uso de un entorno de evaluación se han probado las funcionalidades y características del sistema

directamente en la *FPGA*. Esto permite validar completamente su funcionamiento y descartar fallos de temporización.

Una vez finalizado este proyecto se abren varias líneas de investigación, entre las que se encuentran:

- Pruebas con pacientes en entornos reales. Esto permitirá validar el sistema con nuevas migrañas, de forma que se pueda optimizar el sistema y sacar conclusiones de los resultados obtenidos.
- Ensayos con métodos alternativos de predicción. De esta forma se podrán buscar modelos más precisos o reforzar la efectividad actual del sistema (combinando varios modelos). Entre ellos, se ha considerado el uso de evolución gramatical[15].
- Sustitución de modelos de predicción en caliente (*hot swapping*). Esto permitiría calcular los modelos en la nube y sustituirlos en la *FPGA* sin ser necesario un reinicio del mismo. Para ello, se podrían recibir las actualizaciones en un dispositivo móvil (personalizadas para cada paciente y alimentadas con los nuevos datos recogidos) y cargarlas en el sistema mediante una conexión *Bluetooth*.

Conclusions

A migraine prediction system has been presented in this project. This system has been implemented in a *FPGA* using *VHDL*. That implementation starts from a simulation environment specified in a formal language (*DEVS*). That allows to validate the system in advance and favors the development.

The system manages four biomedical variables that are obtained through various sensors. Two of these four sensors have been conected to an A/D converter to adapt the signal. *I2C* and serial communications are used to transmit information between these elements. The quality of the signals is monitored. In this way, a failure in the set of sensors may be detected. When a failure is detected, the signal is regenerated by means of recovery techniques if necessary. Subsequently, with this information, several migraine predictions are generated based on state-spaces models. Moreover, the models can be changed on real time in case of having a different set of variables. That occurs when the signals that are producing valid data change. In addition, the system is designed to be able to add different prediction mechanisms- As a result, estimations can be generated with different methods, improving the quality of the prediction.

The system has been validated in two different ways. On the one hand, the software simulation of the different modules of the system offers a reliable view of its hardware implementation. After that, through the use of a verification environment, the functionalities of the system have been tested directly on the *FPGA*. This allows to fully verify its operation and rule out timing failures.

Once this work is done, several lines of research are opened, among which are:

- Tests with patients in real environments. This will allow to validate the system with new migraines, in order to optimize the system and draw conclusions from the results obtained.
- Tests with alternative methods of prediction. In this way, it will be possible to

look for models with greater suitability or to reinforce the current effectiveness of the system (combining several models). Among them, the use of grammatical evolution has been considered[15].

- Hot swapping of prediction models. This would allow to calculate the models in the cloud and replace them in the system without needing a restart of the same. To do this, you could receive updates on a mobile device (customized for each patient and fed with the new data collected) and upload them to the *FPGA* through a *Bluetooth* connection.

Bibliografía

[1] Death by Logic. Vhdl: Standard fifo. http://www.deathbylogic.com/2013/07/vhdl-standard-fifo/ (accedido el 22 de enero de 2018).

[2] EP Calandre, J Hidalgo, JM García-Leiva, and F Rico-Villademoros. Trigger point evaluation in migraine patients: an indication of peripheral sensitization linked to migraine predisposition? *European journal of neurology*, 13(3):244–249, 2006.

[3] XESS Corp. Xula2-lx9. http://www.xess.com/shop/product/xula2-lx9/ (accedido el 2 de febrero de 2018).

[4] Hans-Christoph Diener, David W Dodick, Peter J Goadsby, Richard B Lipton, Jes Olesen, and Stephen D Silberstein. Chronic migraine—classification, characteristics and treatment. *Nature Reviews Neurology*, 8(3):162–171, 2012.

[5] Digilent. Digilent pmod ad2: four channel, 12-bit analog-to-digital converter. https://reference.digilentinc.com/reference/pmod/pmodad2/start (accedido el 10 de enero de 2018).

[6] Digilent. Zybo reference manual. https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual (accedido el 21 de enero de 2018).

[7] NJ Giffin, L Ruggiero, RB Lipton, SD Silberstein, JF Tvedskov, J Olesen, J Altman, Peter J Goadsby, and A Macrae. Premonitory symptoms in migraine an electronic diary study. *Neurology*, 60(6):935–940, 2003.

[8] William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

[9] M Linde, A Gustavsson, LJ Stovner, TJ Steiner, J Barré, Z Katsarava, JM Lainez, C Lampl, M Lantéri-Minet, D Rastenyte, et al. The cost of headache disorders in europe: the eurolight project. *European journal of neurology*, 19(5):703–711, 2012.

[10] MikroElektronika. Introduction to fir filters. https://www.mikroe.com/ebooks/digital-filter-design/introduction-fir-filter (accedido el 2 de febrero de 2018).

[11] NandLand. Uart, serial port, rs-232 interface. https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html (accedido el 22 de enero de 2018).

[12] MA Natick. Matlab toolbox, s.i. version 7.14.0.739 (r2012a). https://es.mathworks.com/products/sysid.html, 2010.

[13] NONIN. Nonin medical: Pulse oximeters, regional oximetry, capnography. http://www.nonin.com/ (accedido el 10 de enero de 2018).

[14] NONIN. Oem iii module: Internal pulse oximetry at its best. http://www.nonin.com/OEM-III-Module (accedido el 10 de enero de 2018).

[15] Michael O'Neil and Conor Ryan. Grammatical evolution. In *Grammatical Evolution*, pages 33–47. Springer, 2003.

[16] World Health Organization. Headache disorders, 2006. [Accedido el 08-12-2017].

[17] Josué Pagán, José M Moya, Saurabh Mittal, and José L Ayala. Advanced migraine prediction simulation system. In *Proceedings of the Summer Simulation Multi-Conference*, page 24. Society for Computer Simulation International, 2017.

[18] Josué Pagán, De Orbe, M Irene, Ana Gago, Mónica Sobrado, José L Risco-Martín, J Vivancos Mora, José M Moya, and José L Ayala. Robust and accurate modeling

approaches for migraine per-patient prediction from ambulatory data. *Sensors*, 15(7):15419–15442, 2015.

[19] Jelena M Pavlovic, Dawn C Buse, C Mark Sollars, Sheryl Haut, and Richard B Lipton. Trigger factors and premonitory features of migraine attacks: summary of studies. *Headache: The Journal of Head and Face Pain*, 54(10):1670–1679, 2014.

[20] PLUX. Plux-wireless biosignals website. http://www.biosignalsplux.com (accedido el 10 de enero de 2018).

[21] Rasmussen. Documentation for gpml matlab code. http://www.gaussianprocess.org/gpml/code/matlab/doc/ (accedido el 20 de enero de 2018).

[22] Birthe Krogh Rasmussen and Jes Olesen. Migraine with aura and migraine without aura: an epidemiological study. *Cephalalgia*, 12(4):221–228, 1992.

[23] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[24] MICHAEL BJØRN RUSSELL, BIRTHE KROGH RASMUSSEN, PER Thorvaldsen, and JES Olesen. Prevalence and sex-ratio of the subtypes of migraine. *International journal of epidemiology*, 24(3):612–618, 1995.

[25] Stephen D Silberstein, Richard B Lipton, and Donald J Dalessio. *Wolff's headache and other head pain*. Oxford University Press, 2001.

[26] J Waelkens. Dopamine blockade with domperidone: bridge between prophylactic and abortive treatment of migraine? a dose-finding study. *Cephalalgia*, 4(2):85–90, 1984.

[27] Xilinx. Ise webpack design software. https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html (accedido el 22 de enero de 2018).

[28] Bernard P Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000.

Apéndice A

Representación de números decimales en VHDL

Para la construcción del sistema de predicción se ha hecho necesario tratar con números decimales. Sin embargo, VHDL no cuenta con tipos por defecto que permitan representar este tipo de información (y sean posteriormente sintetizables). Por tanto, se han estudiado diversas opciones, tanto para la representación de números en coma flotante como en coma fija.

A.1. Coma flotante

Los números en coma flotante representan el número decimal dividiéndolo en mantisa, exponente y signo. En concreto, el estándar IEEE754[8] (de aritmética en coma flotante con precisión simple) codifica los números en 32 bits. De ellos, 23 se usan para representar la mantisa (M), 8 para representar el exponente (E) y 1 es reservado para el signo (S) (representado en la figura A.1). Por tanto, este estándar puede representar valores desde −3,4028235e38 hasta 3,4028235e38.

De esta forma, para codificar un número en coma flotante se debe seguir el siguiente procedimiento:

1. Convertir el número a binario. Ej: $5678_{10} = 1011000101110_2$.

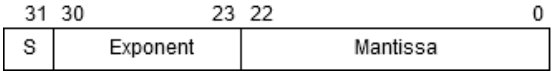


Figura A.1: Estructura de un número en coma flotante mediante el estándar de precisión simple (IEEE754)

2. Expresar el número resultante en la siguiente forma: 1,x...x * 2^n. Ej: $1011000101110_2 = 1,011000101110 * 2^{12}$.
3. Determinar el bit de signo (0 si el número original es positivo, 1 si es negativo). Ej:$5678_{10} \rightarrow S = 0$.
4. Determinar el exponente (exceso 127). Por tanto, el exponente resultante será $E = 127 + n$ (siendo n el exponente obtenido en el paso dos). Si es necesario se añadirán ceros a la derecha hasta completar los ochos bits. Ej: $n = 12 \rightarrow E = 12 + 127 = 139 = 10001011_2$.
5. Determinar la mantisa. Se obtiene completando la obtenida en el paso 2 (x...x) con ceros a la derecha hasta conseguir 23 bits. Ej: $1,011000101110 * 2^{12} \rightarrow M = 00000000000011000101110_2$.

De forma análoga, una vez codificado, se puede obtener de nuevo el valor en base 10 mediante la siguiente expresión:

$$V = (-1)^S * 2^{(E-127)} * (1 + M)$$

En las primeras iteracciones del proyecto se usó este tipo de codificación, usando el paquete *float_pkg* de la librería *IEEE_PROPOSED*. Sin embargo, debido a los costes de las operaciones usando este formato y los altos tiempos de síntesis hicieron que se sustituyerá en adelante y hasta el final del proyecto por una codificación en coma fija.

A.2. Coma fija

En las representaciones de decimales con coma fija se establecen manualmente un determinado número de bits para representar la parte entera y otra cantidad de bits

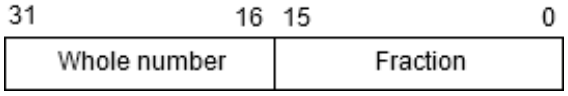


Figura A.2: Ejemplo de estructura de un número en coma fija (32 bits)

para representar la parte fija (pudiendo reservar un bit adicional para representar el signo). En la figura A.2 se puede observar la estructura de un número sin signo codificado en coma fija, usando 16 bits tanto para la parte entera como para la decimal.

De esta forma, asumiendo N bits para la parte entera y M para la decimal el rango de posibles valores a representar va de $-(2^N - 1)$ a $(2^N - 1)$, con un error máximo de $(2^{-M})/2$ en la parte decimal (la mitad del mínimo valor representable con M bits en la parte decimal).

Esta codificación, debido a su simplicidad, tiene un rendimiento notablemente mejor y necesita menos recursos para ser sintetizdo. Sin embargo, limita el rango de valores a usar y puede causar desbordamiento durante la realizacion de las operaciones del sistema (por lo que puede ser necesario usar variabales de mayor tamaño para realizar operaciones intermedias). Para evitar los desbordamientos se calculan los valores máximos que se pueden producir en las operaciones intermedias y se usan estructuras que puedan contenerlos.

Apéndice B

XuLA2-LX9

La placa de desarrollo XuLA2-LX9 (figura B.1) cuenta con una *FPGA* Spartan 6. Es compatible con el entorno de desarrollo Xilinx ISE y cuenta con las siguientes características:

- 32 MB de SDRAM.
- 8 Mb de memoria Flash.
- Encaje para tarjetas microSD.
- Interfaz de 40 pines.
- Oscilador de 12 MHz.
- Microcontrolador PIC 18F14K50.

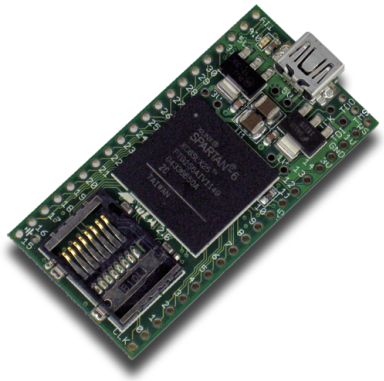


Figura B.1: Placa de desarrollo XuLA2-LX9

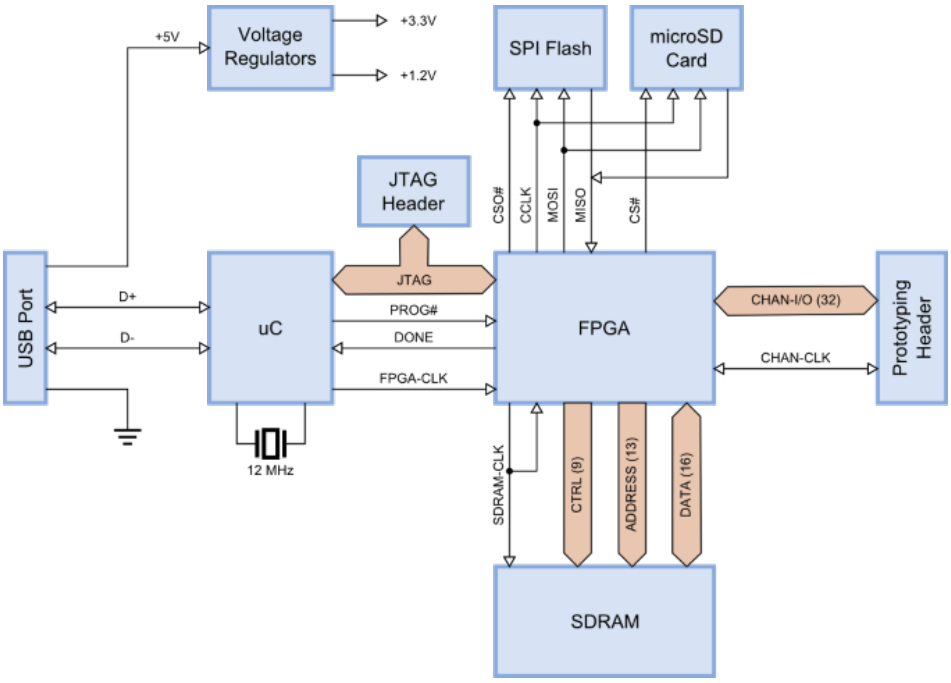


Figura B.2: Distribución de los componentes de la placa XuLA2-LX9

- Puerto USB 2.0.

La distribución de estos elementos puede verse en la figura B.2.

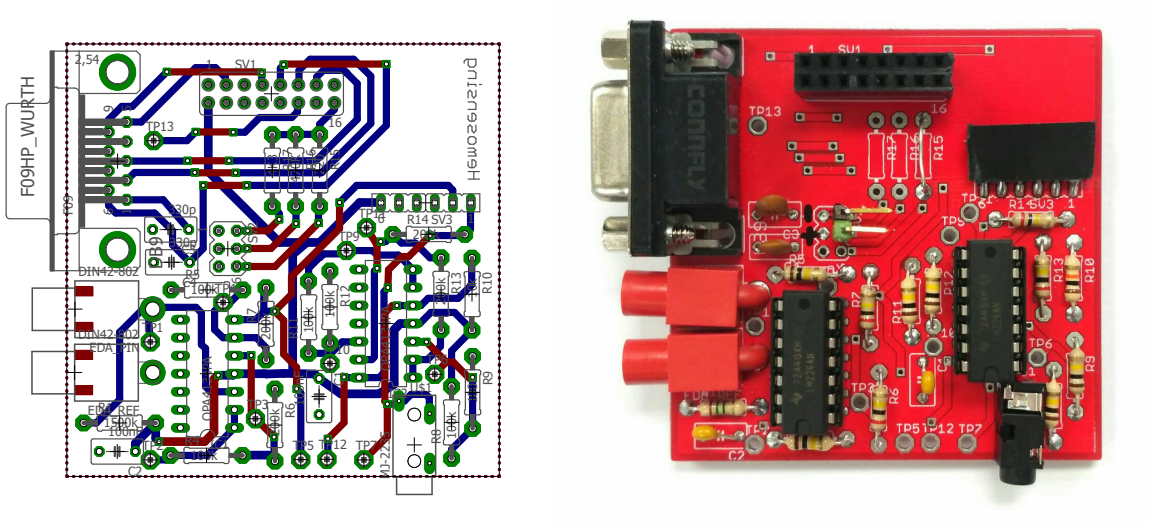
Cabe destacar que, aunque la frecuencia de operación principal es de 12MHz, la placa contiene además *DCMs* (Digital Clock Managers) que permiten aumentar esta frecuencia hasta 384 MHz (32 veces más).

Apéndice C

Esquemáticos de la placa de integración

En la figura C.1 se comparan las conexiones del circuito de la placa de integración de los sensores con el resultado final de la misma.

En la figura C.2 se puede observar el esquemático del circuito. En ella se aprecian las conexiones asociadas a los distintos conectores, el puente de Wheatstone y diversos filtros para estabilizar tanto las señales de sudoración como las de temperatura.



(a) Conexiones de la placa

(b) Placa montada

Figura C.1: Placa para la gestión de sensores

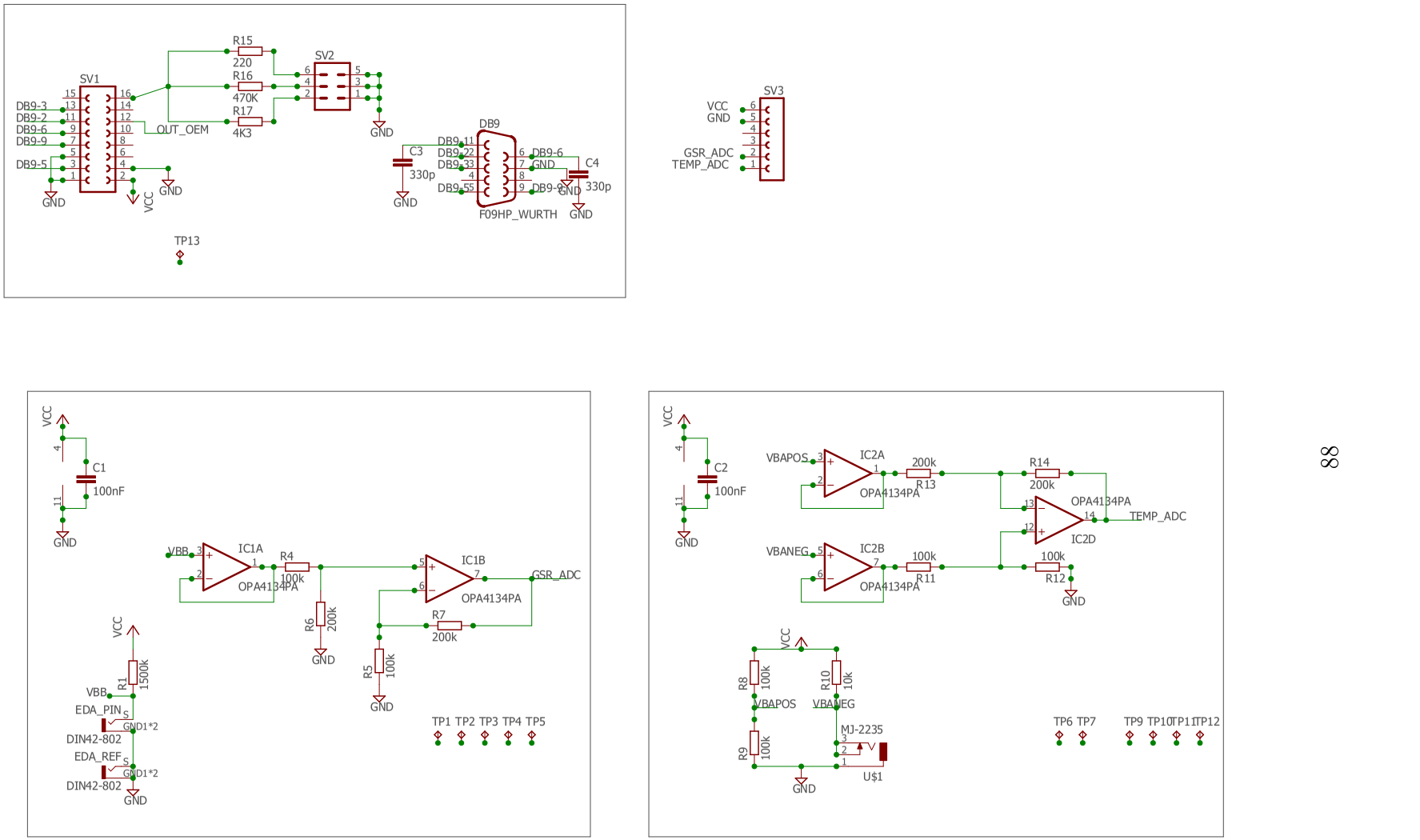


Figura C.2: Esquemáticos de la placa interfaz para la gestión de los sensores